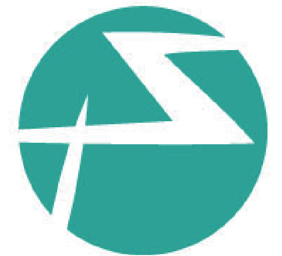

TRON Safe Kernel WG Report

December 2017

TRON Forum

http://www.tron.org/

Copyright © 2017 by TRON Forum

TRON Safe Kernel WG Report (Ver.1.00.00)

You should not transcribe the content, duplicate a part of this document, etc. without the consent of TRON Forum.
For improvement, etc., information in this document is subject to change without notice.
For information about this document, please contact the following:

TRON Forum Secretariat Office
In YRP Ubiquitous Networking Laboratory,
SEIJITSU BLD-1, 2-12-3 Nishi-Gotanda,
Shinagawa, Tokyo 141-0031 Japan
TEL: +81-(3)-5437-0572 FAX: +81-(3)-5437-2399
E-mail: office@tron.org

- Table of contents -

1. Overview of TRON Safe Kernel 3
1.1 Positioning of TRON Safe Kernel 3
1.2 Policy of TRON Safe Kernel 3
1.3 Safety requirements of TRON Safe Kernel 3
1.4 Applicable Hardware 4
2. TRON Safe Kernel Concepts 5
2.1 Basic Terminology 5
2.2 Software and Domain 6
2.2.1 Safety Software and Normal Software 6
2.2.2 Application and System Software 6
2.2.3 Domain 6
2.2.4 Protection among Domains 8
2.3 Memory and Protection Level 9
2.3.1 Memory 9
2.3.2 Protection Levels 9
2.4 API 11
2.4.1 API Overview 11
2.4.2 Safety API 11
2.4.3 Normal API 12
2.4.4 Initial definition API 12
2.4.5 T-Kernel2.0 Compatible API 12
2.5 Execution of Software 13
2.5.1 Execution Unit Overview 13
2.5.2 Execution Priority 14
2.5.3 Consecutive Execution Time 15
2.6 Kernel Object 17
2.6.1 Kernel Object Overview 17
2.6.2 Create a Kernel Object and Delete 17
2.6.3 Object ID and Object Name 17
2.6.4 Protection among Domains for Kernel Object 18
2.6.5 Control Block 18
2.7 System Software 19
2.7.1 Configuration of System Software 19
2.7.2 User-defined System Software 21
2.7.3 State of TRON Safe Kernel 22
3. TRON Safe Kernel/OS Functions 25

3.1 Task Management Functions 25
3.1.1 Overview of Task Management Functions 25
3.1.2 Task Attributes 26
3.1.3 Task State 27
3.1.4 Main Function for Task 29
3.1.5 Task Stack 29
3.1.6 Task Priority 29
3.1.7 Task Scheduling Rule 30
3.1.8 API for Task Management 31
3.2 System State Management Functions 32
3.2.1 Task Scheduling 32
3.2.2 Reference System Information 33
3.2.3 API for System Management 35
3.3 Synchronization and Communication Function 36
3.3.1 Semaphore 36
3.3.2 Event Flag 37
3.3.3 Mutex 38
3.3.4 Message Buffer 40
3.4 Time Management Functions 41
3.4.1 Overview of Time Management Functions 41
3.4.2 Overview of Time Event Handler 42
3.4.3 Attributes for Time Event Handler 43
3.4.4 State of Time Event Handler 43
3.4.5 Main Function for Time Event Handler 45
3.4.6 Scheduling Rule for Time Event Handler 45
3.4.7 Operation of Cyclic Handler 46
3.4.8 Alarm Handler 46
3.4.9 API for Time Management Functions 47
3.5 System Configuration Information Management Functions 48
3.5.1 Overview of System Configuration Information Management Functions 48
3.5.2 Format of the System Configuration Information 48
3.5.3 Standard System Configuration Information 49
3.5.4 API for System Information Management Functions 49
3.6 Domain Management Functions 50
3.6.1 Overview of Domain Management 50
3.6.2 Domain Attribute 51
3.6.3 Domain ID and Priority Order 51
3.6.4 Create a Domain and Execute 51
3.6.5 Exit a Domain 52
3.6.6 Stop a Domain Forcibly 53

3.6.7 State of Domain 53
3.6.8 Disable Dispatching 55
3.6.9 Domain Protection on the Space Basis 55
3.6.10 Domain Protection on the Time Basis 56
3.6.11 API for Domain Management Functions 57
4. TRON Safe Kernel/SM Functions 58
4.1 Device Management Functions 58
4.1.1 Device Management Functions Overview 58
4.1.2 Device Driver Attributes 59
4.1.3 Unit and Subunit 60
4.1.4 Device Name 60
4.1.5 Device Driver Interface 61
4.1.6 Input/output Data 62
4.1.7 Device Event Notification 62
4.1.8 API for Device Driver Management Functions 64
4.2 Interrupt Management Functions 65
4.2.1 Overview of Interrupt Management Functions 65
4.2.2 Overview of Interrupt Handler 66
4.2.3 Interrupt Handler Attributes 67
4.2.4 Execution Priority of an Interrupt Handler 67
4.2.5 Execution Program for Interrupt Handler 67
4.2.6 Stack for Interrupt Handler 67
4.2.7 Maximum Continuous Execution Time for Interrupt Handler 68
4.2.8 API for Interrupt Management Functions 68
4.3 Subsystem Management Functions 69
4.3.1 Overview of Subsystem Management Functions 69
4.3.2 Configuration of Subsystem 69
4.3.3 Information of Subsystem 70
4.3.4 Subsystem Attributes 71
4.3.5 Extended SVC Handler 72
4.3.6 Subsystem Interface 73
4.3.7 API for Subsystem Management Functions 73
4.4 Failure Diagnosis Management Functions 74
4.4.1 Overview of Failure Diagnosis Management Functions 74
4.4.2 Overview of Failure Diagnosis Handler 74
4.4.3 Attributes of Failure Diagnosis Handler 75
4.4.4 State of Failure Diagnosis Handler 76
4.4.5 Start of Failure Diagnosis Handler 77
4.4.6 Execution Program for Failure Diagnosis Handler 77
4.4.7 Time Protection of Failure Diagnosis Handler 78

4.4.8 Available API by Failure Diagnosis Handler 78
4.4.9 Stack for Failure Diagnosis Handler 78
4.4.10 APIs for Failure Diagnosis Management Functions 78
5. Common Rules on TRON Safe Kernel 79
5.1 Common Data Types on TRON Safe Kernel 79
5.1.1 General Data Types 79
5.1.2 Data Type whose meaning is defined 80
5.2 Common Constants for TRON Safe Kernel 81
5.2.1 General Constants 81
5.2.2 Symbol Definition 81
5.3 Error Code 82
5.3.1 Overview of Error Code 82
5.3.2 List of Error Codes 82
5.4 General Specification of TRON Safe Kernel API 86
5.4.1 API Interface Format 86
5.4.2 API Parameters 86
5.4.3 Timeout for API 86
5.4.4 Absolute Time and Relative Time 87
5.4.5 General Specification for Safety API 88
5.4.6 General Specification for Normal API 88
5.4.7 Specific Specifications for API 88
6. Safety Functions 89
6.1 Overview of Safety Functions 89
6.2 Abnormal Exception Functions 90
6.2.1 Overview of Abnormal Exception Functions 90
6.2.2 Kinds of Abnormal exception 91
6.2.3 Overview of Abnormal exception handler 102
6.2.4 Attribute of Abnormal exception handler 103
6.2.5 Register of Abnormal exception handler 103
6.2.6 Execution program for Abnormal exception handler 104
6.2.7 Available API by Abnormal exception handler 105
6.2.8 Stack for Abnormal exception handler 105
6.2.9 API for Abnormal exception functions 105
6.3 Failure Diagnosis Functions 106
6.3.1 Self-diagnosis of kernel 106
6.3.2 Hardware failure diagnosis 107
6.4 Division of domain 108
6.4.1 Spatial separation of the domain 108
6.4.2 Temporal separation of Domain 109
6.5 Protection of setting of System software 110

6.6 Transition to the safety state ... 110
6.7 Guarantee of response performance ... 111
6.8 Debug functions ... 111
7. Configuration of TRON Safe Kernel ... 112
7.1 Overview of Configuration ... 112
7.1.1 Configuration Information ... 112
7.1.2 Initial definition API ... 112
7.1.3 Execution of Configuration ... 113
7.2 System Management Information ... 114
7.3 Memory Management Information ... 120
7.3.1 Overview of memory area ... 120
7.3.2 Memory-map Management Information ... 121
7.3.3 Stack Management Information ... 123
7.3.4 Message Buffer Management Information ... 124
7.4 Domain management information ... 127
7.4.1 Overview of domain management information ... 127
7.4.2 System domain management information ... 129
7.4.3 Safety domain management information ... 130
7.4.4 Normal domain management information ... 133
7.5 Registeration information of User-defined system software ... 135
7.5.1 Initial definition API of registration information of user-defined system software ... 135
8. API specification ... 144
8.1 Difference between Safety API and Normal API ... 144
8.2 API for TRON Safe Kernel/OS ... 144
8.2.1 API for Task Management Functions ... 144
8.2.2 API for System State Management Functions ... 180
8.2.3 API of the Synchronous/Communication Functions (Semaphore) ... 190
8.2.4 API for Synchronization and Communication Functions (Event Flag) ... 200
8.2.5 Synchronization and Communication Functions (Mutex) ... 211
8.2.6 API for Synchronization and Communication Functions (Message Buffer) ... 221
8.2.7 API for Time Management Functions ... 232
8.2.8 API for System Configuration Information Management ... 252
8.2.9 API for Domain Management Functions ... 256
8.3 API for TRON Safe Kernel/SM ... 268
8.3.1 API for Subsystem Management Functions ... 268
8.3.2 API for Device Management Functions ... 271
8.3.3 API for Interrupt Management Functions ... 296
8.3.4 API for Failure Diagnosis Functions ... 300
8.3.5 API for Abnormal Exception Functions ... 304
9. Appendix ... 309

9.1 The background to designing the specification 309

Description format of this specification document

The description format of this specification document is shown below.

■ Supplement

The supplement of the specification is described in the text of the specification document. The supplement is shown by the item name surrounded by "[ ]"

The supplementary explanation to understand the specification is described at the item of "[Supplement explanation]".
The functional restriction of TRON Safe Kernel created by the hardware functional restriction is described at the item of "[Hardware restriction]"

■ Description format of explanation of API

The explanation of API in the specification document consists of the following items. All items may not be described at all APIs.

Explanation of API

The name and explanation of API are described.

Interface of C language

[Safety API]

Interface of C language of Safety API is described.

[Normal API]

Interface of C language of Normal API is described.

[Initial definition API]

Interface of C language of Initial definition API is described.

Parameter

This section explains the information which is passed to TRON Safe Kernel when API is invoked.

Return parameter

The section explains the information which is returned by TRON Safe Kernel when the execution of API ends.
The return parameter which is returned as a function value of API is called "return value".
The return parameter consists of "return value" and the one which returns the information to the reference address of pointer passed as a parameter.

Error code

This section explains the error which may occur at API.

Valid Context

The context which can invoke API is described. The context is divided shown as below.

Task portion:	Task portion or quasi-task portion is running
Time event handler:	Time event handler (Periodical handler or alarm handler) is running
Task independent portion:	The portion except for task portion, quasi-task portion and time event handler is running

The meaning of the symbols indicating possibility to invoke API is shown as below.

"○" The API can be executed in the context.

"×" Error E_CTX occurs if the API is executed in the context.
(If the return parameter of API cannot return E_CTX, the processing is described at the explanation of each API)

Description

This section explains the function of API.

This section explains each API. This section does not explain the common specification of API (parameter packet or error code and so on).

When this document shows the Safety API (ts_xxx_vvv) and the Normal API (tn_xxx_vvv), it indicates "ts_xxx_yyy/tn_xxx_yyy".

In case of selecting and setting some values as the parameter, this document explains the specification by the description method shown below.

(x || y || z)

Select and specify one of x or y or z

x | y

It is possible to specify x and y at the same time.(In case of specifying at the same time, take the sum of x and y)

[x]

x may or may not be specified.

Description example of parameter

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

In case of "wfmode := (TWF_ANDW || TWF_ORW) | [TWF_CLR]" ,the setting of "wfmode" becomes any of four parameters below.

TWF_ANDW

TWF_ORW

TWF_ANDW | TWF_CLR

TWF_ORW | TWF_CLR

[Supplement]

This section describes the additional notes of the explanation.

1. Overview of TRON Safe Kernel

1.1 Positioning of TRON Safe Kernel

Real time OS for an embedded system has been specified by TRON project, such as T-Kernel2.0 (released in 2011), T-Kernel, uITRON and so on.

In the future, functional safety will become more important in the embedded software, therefore TRON Safe Kernel is specified as OS with functional safety, and developed from the conventional TRON OS.

1.2 Policy of TRON Safe Kernel

Policy of TRON Safe Kernel is indicated as follows.

(1) TRON Safe Kernel realizes functional safety as an OS and provides safety functions for software running on an OS on the assumption that it is used as an OS for a high-level functional safety software.
Safety integrity level is based on SIL3 in IEC61508.

(2) TRON Safe Kernel can execute user programs that meet a target function safety level. It also provides a function that allows for an easy implementation of a user program.

(3) TRON Safe Kernel can execute user programs that do not meet a function safety level. It provides the functions same as T-Kernel2.0 to a level where the policy (1) and (2) can be realized.

(4) TRON Safe Kernel can execute user programs in different functional safety levels independently. These programs are assumed to be user programs on the basis of the specific functional safety level of (2) and the ones on the basis of other than the specific functional safety level of (3).

1.3 Safety requirements of TRON Safe Kernel

Safety requirements of functional safety on TRON Safe Kernel are indicated as follows.

(1) Functional safety of OS itself

TRON Safe Kernel is developed on SIL3 in IEC61508. OS specifications shall be met it.

(2) Functions for basic safety

TRON Safe Kernel has basic safety functions for detecting an abnormal operation on hardware and software and transiting to safe state for OS. TRON Safe Kernel has a function for executing self-diagnosis for failure detection. These functions are realized as an abnormal exception function and a failure diagnosis function.

(3) Safety function for an application with functional safety

TRON Safe Kernel has a function executing applications on the basis of the specific safety level and on other than the specific

safety level, realized as a domain management function.

1.4 Applicable Hardware

TRON Safe Kernel can realize all the functions in the standard hardware as follows. If a part of the hardware function is lacked, some restriction occurs at the function of TRON Safe Kernel. The concrete restriction is described at "Hardware restriction"

(1) Processor

Hardware for executing a program.

An execution unit of single program in a processor is called 'Processor core' or 'Core'. A processor composed of multi core is called 'Multi core processor'.

TRON Safe Kernel is based on single core processor, not multi-core processor.

A processor has two or more step execution modes such as privileged mode non-privileged mode. In non-privileged mode, access from a program to a peripheral device can be prohibited.

(2) Memory

Memory allocated physically on an address space in processor.

It is necessary for a processor to have a hardware function for protecting the specific memory area from software, such as MMU (Memory Management Unit), MPU (Memory Protection Unit) and so on. And a processor in the execution mode of the program can realize the memory area which can be accessed by only Privileged mode.

(3) System timer

Hardware timer used for measuring (a period of) time and time.

System timer can generate a timer interrupt periodically by the determined time interval or at the specified time.

(4) Highest priority timer

Timer which can generate higher priority interrupts than other interrupts which system software uses. Or it can generate interrupt even if the other interrupts which system software uses are masked. When system software cannot be executed property, the highest priority timer is used as a watch dog to detect that state.

(5) Peripheral device

Hardware controllable by program, built in processor or connected externally.

A peripheral device is not controlled by TRON Safe Kernel directly. It is controlled by user-defined software such as device driver.

2. TRON Safe Kernel Concepts

2.1 Basic Terminology

Basic terms in the specifications are defined as follows.

(1) Implementation-dependent and implementation-defined

' Implementation-dependent ' means an item in which the behavior of something varies according to the hardware which TRON Safe Kernel works or system operating conditions. The specifications of implementation-dependent should be defined for each implementation. This is called 'Implementation-defined '. The specifics of the implementation should be described clearly in the implementation specifications.

(2) Configuration and Configuration definition file

Various setting about system of TRON Safe Kernel and to define software embedded in initial state is called 'Configuration'. The information about configuration is described in ‘Configuration definision file’. It is NOT possible for the information about configuration to change by executing a program.

[Supplement]

For implementation-defined, it is necessary to change a program code, while, for the configuration, it is necessary to indicate a setting value, a data like initial value and system software embedded in initial state, not to change a program code.

(3) Interrupt and abnormality exception

'Interrupt and Exception' means an operation started up other program on executed program for the event detected by processor. In TRON Safe Kernel, an operation started by the event difficult to continue an execution such as illegal addressing, undefined instruction executing, etc. is called 'Abnormality exception', others 'Interrupt'. Applicable events for the abnormality exception and the interrupt are implementation-defined'.

[Supplement]

To distinguish 'Interrupt' and 'Exception' depends on a processor specification. To avoid confusion in TRON Safe Kernel, the name of ‘Exception’ is used only as ‘Abnormal exception’ in this specification. General interrupt and exception specified by processor specification etc. are collectively called 'Interrupt'.

(4) Context

'Context' means an environment for executing a program. Same context has equivalent address space and common stack area in this specifications.

2.2 Software and Domain

2.2.1 Safety Software and Normal Software

TRON Safe Kernel has two (2) kinds of software, Safety software and Normal software, on the basis of functional safety.

Safety software has high reliability and safety for functional safety, the others Normal software.

Safety level of safety software is defined by 'Implementation-defined'. Safety software shall be met its safety level.

2.2.2 Application and System Software

TRON Safe Kernel has two (2) kinds of program, application and system software.

Application means user software executing on TRON Safe Kernel created by a user.

System software is composed of TRON Safe Kernel itself and User-defined system software. User system software means a program developed by a user on the basis of user hardware and application and registered to the system software

System software shall be safety software.

Application is Safety software and/or Normal software, and application which consists of Safety software only is possible.

System software is always executed on privileged mode, prior to application. That is, in the execution mode which a processor provides, system software is executed on privileged mode and application is executed on non-privileged mode.

[Hardware restriction]

If a processor does not have plural execution modes of the program such as privileged mode or non-privileged mode, a program consists of system software only and an application cannot exist. The program which user can create is user-defined software only.

2.2.3 Domain

TRON Safe Kernel manages software and its resource by domains. There are several domains in TRON Safe Kernel and software belongs to one of domains.

The safety level of the software in a domain should be same.

Domain for system software is called "System domain". Safety domain in a broad sense is System domain and the domain which Safety software of the application belongs to. The latter is called "Safety domain in a narrow sense" or simply "Safety domain".

A domain to which Normal software belongs is called Normal domain. All the software belonging to Normal domain is application.

. From the above, domains of TRON Safe Kernel are classified into the following types.

(1) System domain

Domain which system software belongs to. An application cannot belong to System domain. System software is Safety software and therefore System domain is Safety domain in a broad sense. There is only one System domain.

(2) Safety domain

Domain which Safety software of an application belongs to. There is one or more Safety domain.

The maximum number of Safety domains is implementation-defined.

(3) Normal domain

Domain for Normal software in application.

Normal domain exists one (1) and over, or not exist, in the latter case, application also not exist. Maximum number of Normal domains is implementation-defined.

Safety domain and Normal domain are collectively called 'Application domain'.

Relations between domain and software are shown in the "Figure 2-1 Relations between domain and software".

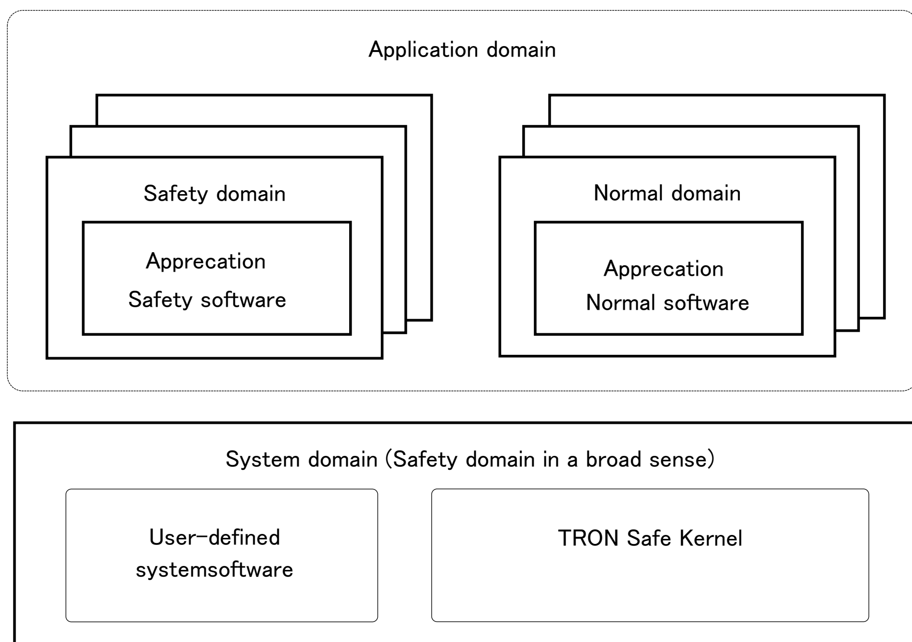

Figure 2-1 Relations between domain and software

[Hardware restriction]

If a processor does not have plural execution modes of the program such as privileged mode or non-privileged mode, a program consists of system software only and domain consists of System domain only. If a processor does not have the function to protect the specific memory area, Application domain consists of one Safety domain only.

2.2.4 Protection among Domains

Software on the domain is protected from the one on other domain in space level and time level on that order basis.

Space level domain protection means a memory protection among domains (Refer to '3.6.9 Domain Protection on the Space ' in details.)

Time level domain protection means a protection of execution time assigned to software. (Refer to 3.6.10 Domain Protection on the Time ' in details.)

Software on the specific domain is protected from the operation by the software on other domain (Refer to '2.6.4 Protection among Domains for Kernel ' in details.)

System software is protected from an application by protection among domains. Safety software is protected from Normal software.

2.3 Memory and Protection Level

2.3.1 Memory

TRON Safe Kernel operates the memory allocated physically on the address space in processor, not virtual memory such as secondary storage and so on.

TRON Safe Kernel has only one address space, NOT possible to multiple memory address space. Memory address is allocated statically, not dynamically.

Memory is divided to memory spaces. Address and size of memory space is set statically. It is possible to set attributions of cache and memory access for each memory respectively. Applicable address, size and attributes are implementation-defined because of depending on the hardware.

Memory space belongs to the specific domain decided statically. It is possible for one (1) domain to have multiple memory spaces. Memory space is protected in space level from accessing by software on other domain.

2.3.2 Protection Levels

Protection level is the general term of execution mode of software on processor and its access protection of memory. It has values between '0' and '3', smaller more privileged.

Execution mode of software on processor is assigned to protection level. Mapping between them is implementation-defined. It is possible for different protection level to assign the same execution mode because of processor not always having four (4) execution modes. It is possible for memory protection such as MMU, etc. and access to peripheral devices to have protection level '0' and '1', not protection level '2' and '3'.

Protection level is defined on each domain respectively. Software and memory has protection level of domain belonged to. System domain has protection level 0, Safety domain 2, Normal domain 3, and Reserved for extension 1.

[Supplement]

T-Kernel specifications specifies that software enables to access to the memory in same protection level or lower. TRON Safe Kernel specifies that protection level is defined on the basis of domain kind so as not to conflict with specifications of T-Kernel. In TRON Safe kernel, it is NOT possible for software with same protection level on other domain to access the memory because of prohibiting memory access among domains. Protection level on TRON Safe Kernel is the general function of memory protection with execution mode of processor (hardware).

It is possible for T-Kernel to set and change a protection level for calling a system call. It is NOT possible for TRON Safe Kernel to change a protection level for calling a system call because it is defined on each domain respectively.

[Hardware restriction]

Many processors have execution mode in two (2) levels such as privileged mode / user mode, protection level '0', and '1' corresponds to privileged mode, protection level '2', and '3' user mode.

Relation among protection level, domain and execution mode in a processor with execution modes in two (2) levels is shown in "Table 2-1 Protection level and Domain".

If a processor does not have plural execution modes of the program, the execution mode of all protection level becomes same. In this case TRON Safe Kernel consists of system software only. Therefore there is only a program whose protection level is "0".

Table 2-1 Protection level and Domain

Protection level	Domain	Execution mode of processor
0	System domain	Privileged mode
1	Reserved	Privileged mode
2	Safety domain	User mode (Non-Privileged mode)
3	Normal domain	User mode (Non-Privileged mode)

2.4 API

2.4.1 API Overview

API (Application Programming Interface) means an interface for application for using functions of system software.

APIs of TRON Safe Kernel consist of Safety APIs used at Safety domain and Normal APIs used at Normal domain.

When the program of Safety domain operate Kernel objects of Normal domain, it uses Safety APIs.

There is Initial definition API as a special Safety API. Initial definition API is used for registration of software embedded in initial state of TRON Safe Kernel.

In Normal domain, T-Kernel2.0 library which has APIs compatible with T-Kernel2.0 is available.

The relation between each API and domains is shown in "Figure 2-2 Relation between APIs and Domains".

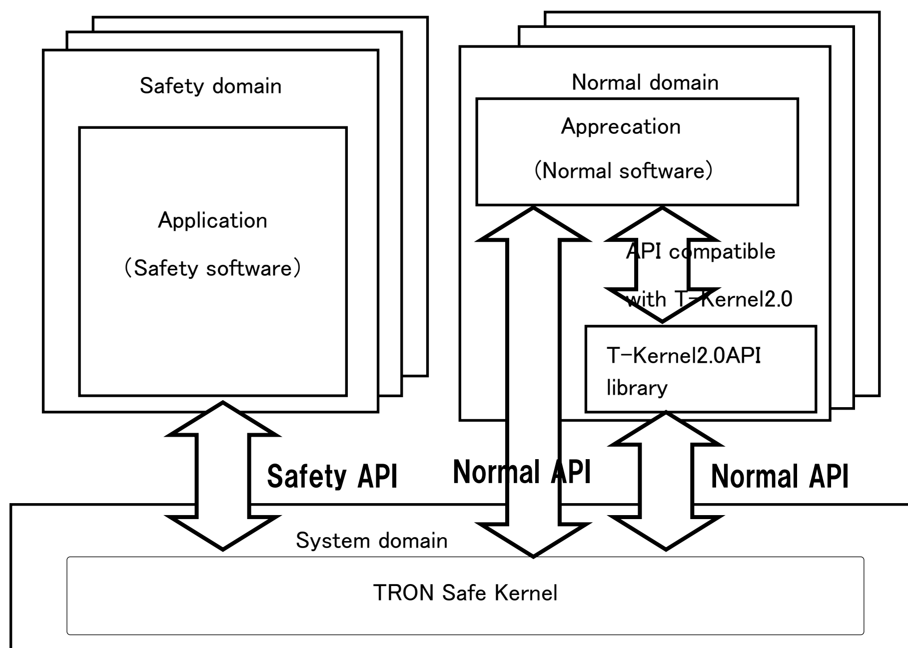

Figure 2-2 Relation between APIs and Domains

2.4.2 Safety API

Safety APIs on TRON Safe Kernel are used for Safety software (software which belongs to System domain or Safety domains.

Safety API is defined as a function in the name with prefix 'ts_' in C language. It calls TRON Safe Kernel immediately after executing. Safety API is operated on TRON Safe Kernel.

For Safety APIs, in additions to Task Management Functions, System State Management Functions, Synchronization and Communication Functions, Time Management Functions, System Configuration Management Functions, Device Management Functions, Subsystem Management Functions, Domain Management Functions(partially) , Interrupt Management Functions, Failure Diagnosis Functions and Abnormal Exception Functions which are not available for Normal APIs are supported. Refer to "8 API specification" in detail.

2.4.3 Normal API

Normal API on TRON Safe Kernel is used for software on Normal domain.

Normal API is defined as a function in the name with prefix 'tn_' in C language. It calls TRON Safe Kernel immediately after executing. Normal API is operated on TRON Safe Kernel.

Normal API has the limited function based on Safety API for using on Normal domain in the basis of Safety API. Normal API is the subset functions of Safety API (except for the part of processing. Detail is shown in "8 API specification").

For Normal APIs, Task Management Functions, System State Management Functions, Synchronization and Communication Functions, Time Management Functions, System Configuration Information Management Functions, Device Management Functions, Subsystem Management Functions, Domain Management Functions (partially) are supported. Refer to "8 API specification" in detail.

2.4.4 Initial definition API

Initial definition API is the API of TRON Safe Kernel used at initializing process of TRON Safe Kernel.

Initial definition API is defined as a function of C language whose name begins at "TS". Initial definition API can be described in the defined place of configuration definition file only, and Initial definition API is executed at only initialization processing of TRON Safe Kernel. Initial definition API can not be used in the program described by user. Refer to "7 Configuration of TRON Safe Kernel" in detail.

2.4.5 T-Kernel2.0 Compatible API

T-Kernel2.0 compatible API is the function library for porting an application on conventional T-Kernel2.0 to on TRON Safe Kernel without difficulty. Its specifications are compatible with those of T-Kernel2.0 API, excluding functions difficult to supply on the basis of functional safety.

T-Kernel2.0 compatible API is implemented as T-Kernel2.0 API library. Library functions are functions in C language, executed as a part of program in Normal software. Therefore T-Kernel2.0 API library is not included in the software of TRON Safe Kernel itself and is the positioning of Normal software. However Normal API is called on its execution of library function in some cases.

[Supplement]

In TRON Safe Kernel specification, whether T-Kernel 2.0 API library and T-Kernel 2.0 compatible API are implemented or not and how to implement them

are implementation-defined .

2.5 Execution of Software

2.5.1 Execution Unit Overview

Program execution unit controlled by TRON Safe Kernel is called 'Execution unit'. Program in one (1) execution unit is executed sequentially.

Execution unit for application is composed of 'Task' and 'Time event handler'.

Execution unit for user system software is composed of 'Task', 'Time event handler', 'Interrupt handler', 'Failure diagnosis handler', 'Abnormality exception handler' and 'Extended SVC handler'.

Execution unit is indicated as follows.

(1) Task

Task is an execution unit executed regularly. Conceptually, multiple tasks are executed in parallel on TRON Safe Kernel. "Executing in parallel" means a conceptual behavior viewed from the perspective of software. Actually, multiple tasks are executed on the task scheduling.

Task is a kernel object, and can be created and executed on all domains of System domain, Safety domain and Normal domain.

(2) Time event handler

Time event handler is the execution unit started on the basis of time.

Time event handler has two (2) kinds of unit, 'Cyclic handler' and 'Alarm handler'. Cyclic handler is a time event handler started on the basis of the specific intervals. Alarm handler is a time event handler started once at the specific time.

Time event handler is a kernel object, and can be created and executed on all domains of System domain, Safety domain and Normal domain.

(3) Interrupt handler

Interrupt handler is the execution unit operated at interrupting.

Interrupt handler is executed on System domain as user system software.

(4) Failure diagnosis handler

Failure diagnosis handler is the execution unit operated at specific intervals or specific time for detecting a failure on hardware.

Failure diagnosis handler is executed on System domain as user system software.

(5) Abnormal exception handler

Abnormal exception handler is the execution unit operated if an abnormal exception occurs.

Default abnormal exception handler is registered statically as a part of TRON Safe Kernel

The execution of the abnormal exception handlers is not prevented even if the kernel is interruptdisabled state or dispatch disabled state.

(6) Extended SVC handler

Extended SVC handler calls a function of subsystem, one of elements in user system software.

Extended SVC handler is the execution unit operated if extended SVC on subsystem called by other program. Extended SVC

handler is one of programs composed of subsystem. One (1) subsystem has one (1) extended SVC handler.

2.5.2 Execution Priority

Execution priority is assigned to each execution unit in software respectively. TRON Safe Kernel executes software in the order setting on the basis of its execution priority. This operation is called 'Scheduling'.

Execution priority has two (2) kinds of priority, 'Task priority' and 'System priority'. System priority is prior to task priority.

(1) Task priority

Task priority is the priority assigned to task, time event handler and failure diagnosis handler.

Task priority is an integer starting from '1'. Smaller value has higher priority. For example, task with priority '1' is prior to execute than one with priority '2'.

About maximum value of task priority (low priority), the value between '16' and '250' is set by the configuration (TS_LST_PRI).

And the range of task priority which can be specified is defined on each domain respectively.

The priority of time event handler is the highest in the domain which it belongs to

(2) System priority

System priority is the priority assigned to TRON Safe Kernel itself, Interrupt handler, and Abnormality exception handler.

Value of system priority is negative, '−1' or less. Lower value has higher priority.

About minimum value of system priority (high priority), the value between '−250' and '−1' is set by the configuration (TS_HST_SPRI).

Assignment of system priority is implementation-defined as follows.

Abnormality exception handler (high priority) ＞ TRON Safe Kernel ＞ Interrupt handler (low priority)

It is possible for interrupt handler to assign higher priority than that of TRON Safe Kernel. In this case, applicable API for interrupt handler is limited in some cases. Its limitation is implementation-defined.

2.5.3 Consecutive Execution Time

Consecutive execution time is assigned to each execution unit in software respectively. TRON Safe Kernel operates as an abnormality exception if execution unit is executed after passing the consecutive execution time.

Consecutive execution time for each execution unit is indicated as follows.

(1) Task

Consecutive execution time for task is the time transiting the task from on running to on other state. Consecutive execution time also finishes measuring if execution changed to the task with higher priority.

TRON Safe Kernel notifies an abnormality exception immediately after passing the consecutive execution time for task. The task is suspended its execution and abnormality exception handler is executed.

Time executing the execution unit other than task by interrupt on task executing is excluded from the consecutive execution time, for example, time of executing interrupt handler, time event handler and failure diagnosis handler. In this case, current consecutive execution time is kept and resumed measuring at returning.

(2) Time event handler

Consecutive execution time for time event handler is the time from executing a time event handler to exiting.

TRON Safe Kernel notifies an abnormality exception immediately after passing the consecutive execution time for time event handler. The time event handler is suspended its execution and abnormality exception handler is executed.

Time executing other execution unit by interrupt on time event handler executing is excluded from the consecutive execution time, for example, time of executing Interrupt handler, Time event handler on other domain, Task on other domain and so on. In this case, current consecutive execution time is kept and resumed measuring at returning.

(3) Extended SVC handler

Extended SVC handler is called by task or time event handler and executed on its context. Extended SVC handler has no consecutive execution time independently. Its time is added to the execution time of the caller, task or time event handler.

(4) Interrupt handler

Consecutive execution time for interrupt handler is the time from executing an interrupt handler to exiting.

TRON Safe Kernel notifies an abnormality exception after passing consecutive execution time for interrupt handler. In this case, the interrupt handler is not suspended (different from task) because consecutive execution time is checked at finishing executing. If an interrupt handler executed for a long time by any of the failures, its abnormal operation is detected by failure diagnosis functions.

If other interrupt handler executed by multiple interrupt on the interrupt handler running, its operation time is not added to the consecutive execution time.

(5) Failure diagnosis handler

TRON Safe Kernel notifies an abnormality exception immediately after passing the consecutive execution time for failure diagnosis handler. The failure diagnosis handler is suspended its execution and abnormality exception handler is executed.

If other execution unit is executed by interrupt on the failure diagnosis handler running, its operation time is not added to the consecutive execution time, for example, execution time of interrupt handler, other failure diagnosis handler and so on. In this

case, the consecutive execution time is started to measure again from the stored value after returning to the failure diagnosis handler as caller.

Failure diagnosis handler operates to mask an interrupt in some cases. Passing the consecutive execution is not immediately detected in the period of interrupt masking. In this case, abnormality exception is occurred at unmasking for the interrupt. Failure of the interrupt mask time is detected by a failure diagnosis functions.

Abnormality exception is not occurred on executing a system call, applied to all execution units. If passing the consecutive execution time on executing a system call, abnormality exception is occurred after finishing executing the system call.

If passing the consecutive execution time on executing extended SVC, abnormality exception is occurred.

2.6 Kernel Object

2.6.1 Kernel Object Overview

Software resource managed by TRON Safe Kernel is called 'Kernel object'.

Kernel object means an execution unit such as task, time event handler and other program execution units, and Mutex, message buffer and other synchronization or communication units.

Kernel objects are shown in "Table 2-2 Kernel objects list".

Table 2-2 Kernel objects list

Category	Kind	Remarks	References
Task	Task	Execution unit (Program)	Section 3.1
Time event handler	Alarm handler	Execution unit (Program)	Section 3.4
	Cyclic handler	Execution unit (Program)	
Synchronization and Communication unit	Event flag	—	Section 3.3
	Semaphore	—	
	Mutex	—	
	Message buffer	—	

2.6.2 Create a Kernel Object and Delete

Kernel object is created and deleted dynamically by calling API. Kernel object belongs to the domain for the software calling its API.

As an exception, there is an initial task in each domain. An initial task is the special task executed at first in each domain and belonging domain is decided at constructing system. To create and execute an initial task is excuted by kernel ,so it is not necessary to invoke API expressly.

Resources for kernel object, such as memory and so on, is assigned statically at configuring the system.

So processing in creating Kernel objects is initializing data only.

[Supplement]

A kernel object belonging to a domain is created by the initial task of the domain directly or indirectly.

2.6.3 Object ID and Object Name

Unique ID number is assigned to a kernel object at creating. TRON Safe Kernel identifies a kernel object by the ID number and operates the API for the kernel object on the basis of ID number.

ID number for the kernel object is called on the basis of its kind, for example, ID number for task is called 'Task ID' for semaphore 'Semaphore ID', etc. and for kernel objects collectively called 'Object ID'.

Object ID has a specific value (unique value) for each type of object through all domains. That is, if there are same kind of multiple objects, regardless of whether the domain which those kernel objects belong to are same or different, all kernel object IDs have different values.

Delegated ID number of kernel object can be assigned for new kernel object which will be created later.

Object name for kernel object is the unique name set at creating.

However object name may not be specified at creating. In that case the kernel object does not have object name.

It is possible for object name to set by character string in eight (8) and less, one (1) byte. Same kind of kernel objects on same domain have different object name each other, for example, all tasks on same domain have different object name each other.

Safety API is used for referring to an ID number for kernel object on other domain. In that case object name should be specified at creating.

2.6.4 Protection among Domains for Kernel Object

Kernel object belongs to the specific domain. It is possible to create a kernel object and delete by the software on same domain. Operating a kernel object on other domain is limited as shown below.

(1) It is NOT possible for Normal software to operate a kernel object on other domain. It is possible for Normal API to operate a kernel object on the self domain.

(2) It is possible for Safety software to operate a kernel object on other application domain.

It is NOT possible for Safety software to operate a kernel object on other domain, if its kernel object transited to waiting state by the operation. It is possible to execute polling for the kernel object on other domain.

(3) It is NOT possible to operate a kernel object on system domain by application.

2.6.5 Control Block

TRON Safe Kernel has a data to manage each kernel object. This data is called "control block" of a kernel object. Some kernel objects have not only control block but also execution program and memory area.

A control block is an internal data of TRON Safe Kernel and cannot be referred and operated by the external portion.

[Supplement]

A control block is an internal data of TRON Safe Kernel, so a user does not need to be aware of its existence. However the explanation of a control block is necessary at the failure diagnosis of TRON Safe Kernel at the functional safety, so a control block is described here.

2.7 System Software

2.7.1 Configuration of System Software

System software is composed of TRON Safe Kernel itself and User-defined system software.

TRON Safe Kernel has TRON Safe Kernel/OS, TRON Safe Kernel/SM and TRON Safe Kernel/Core.

(1) TRON Safe Kernel/OS

TRON Safe Kernel/OS manages kernel objects mainly.

The main functions of TRON Safe Kernel/OS are Task management, System state management, Synchronous communication management, Time management, System configuration information management, and Domain management.

It is possible for applications to use the functions of TRON Safe Kernel/OS by API.

(2)TRON Safe Kernel/SM

TRON Safe Kernel/SM manages User-defined system software mainly.

TRON Safe Kernel/SM has functions of Device management, Interrupt management, Sub-system management, Failure detection management and abnormal exception handler.

It is possible for application to use functions on user-defined system software via TRON Safe Kernel/SM.

(3) TRON Safe Kernel/Core

TRON Safe Kernel/Core manages an operation relating to hardware on TRON Safe Kernel.

Main functions of TRON Safe Kernel/Core are functions of Start processing, Low-level memory management, Low-level interrupt management, Low-level time management, and Context management.

TRON Safe Kernel/Core manages the operation depended on hardware in TRON Safe Kernel. TRON Safe Kernel is implemented to the specific processor compatible with TRON Safe Kernel/Core.

It is NOT possible for applications to use the functions of TRON Safe Kernel/Core directly.

Configuration and functions of TRON Safe Kernel are shown in "Table 2-3 Configuration and functions of TRON Safe Kernel".

Table 2-3 Configuration and functions of TRON Safe Kernel

Configuration	Functions	Explanation
TRON Safe Kernel /OS	Task management	Manage a task
	System management	Manage system state such as scheduling for task and so on
	Synchronous and communication management	Manage a synchronous communication function
	Time management	Manage system time and time event handler
	System configuration information management	Manage system configuration information
	Domain management	Manage a domain
TRON Safe Kernel /SM	Device management	Manage a device driver
	Interrupt management	Manage an interrupt handler
	Sub-system management	Manage a subsystem
	Failure diagnosis management	Manage a failure diagnosis handler
	Abnormal exception handler	Handler of abnormal exception
TRON Safe Kernel /Core	Startup processing	Start of system software, execution of applications
	Low-level memory management	Basic control such as setting of memory and cache
	Low-level interrupt management	Reception and control of interrupt from hardware
	Low-level time management	Scheduling processing by using system timer
	Context management	Dispatching processing (Switching processing of task and handler)

2.7.2 User-defined System Software

User-defined system software consists of the following software.

(1) Sub-system

A subsystem is a program to extend the functions of the system software. The interface which calls the functions of a subsystem is called "extended SVC".

A subsystem consists of any number of tasks, time event handlers and an extended SVC handler. And it has a subsystem interface function as an interface of TRON Safe Kernel.

(2) Device driver

A device driver is a program to control a peripheral device.

A device driver consists of any number of tasks, time event handlers. And it has a device interface function as an interface of TRON Safe Kernel.

(3) Interrupt handler

An interrupt handler is an execution unit of a program executed when an interrupt occurs.

An interrupt handler has a close relationship with a normal or particular subsystem or device driver, and it can be considered that an interrupt handler is functionally a part of the subsystem and the device driver.

(4) Failure diagnosis handler

Failure diagnosis handler is an execution unit of a program executed regularly and when it is determined to detect hardware failure.

(5) Abnormal exception handler

An abnormal exception handler is an execution unit of a program executed when an abnormal exception occurs.

A default abnormal exception handler is registered statically as a part of TRON Safe Kernel.

An execution of an abnormal exception handler is not inhibited even if the kernel is interrupt disabled state or dispatch disabled state.

[Supplement]

TRON Safe Kernel does not define standard subsystem, device driver, interrupt handler and so on. However it may define subsystem, device driver, interrupt handler and so on in the specific system. They are implementation-defined.

2.7.3 State of TRON Safe Kernel

The state of TRON Safe Kernel is roughly classified as follows.

(1) Kernel starting state (Kernel stopped state)

In this state TRON Safe Kernel is started by power on or system reset and so on. In this state the specified starting process is expected by the starting processing program of TRON Safe Kernel, and the state transits to "Kernel initializing state" after the processing ends. In this state TRON Safe Kernel is not running and all the functions of this software cannot be used. Therefore this state is also called "Kernel stopped state".

(2) Kernel initializing state

In this state, TRON Safe Kernel is activated and initializing the kernel based on the information of the configuration.
After the initializing processing, it starts the execution of system domain and specific domain and transits to "Kernel running state". The Safety domain starting execution is defined statically.
In this state only an abnormal exception processing works. If an abnormal exception is detected, the kernel state transits to "Kernel error state".

(3) Kernel running state

The kernel is running normally. All the functions of TRON Safe Kernel are available.

(4) Kernel error state

Abnormality is detected and an abnormal exception occurs and the processing corresponding to it is executing.

The summary of state transition of TRON Safe Kernel is shown in "Figure 2-3 State transition of TRON Safe Kernel (Summary)".

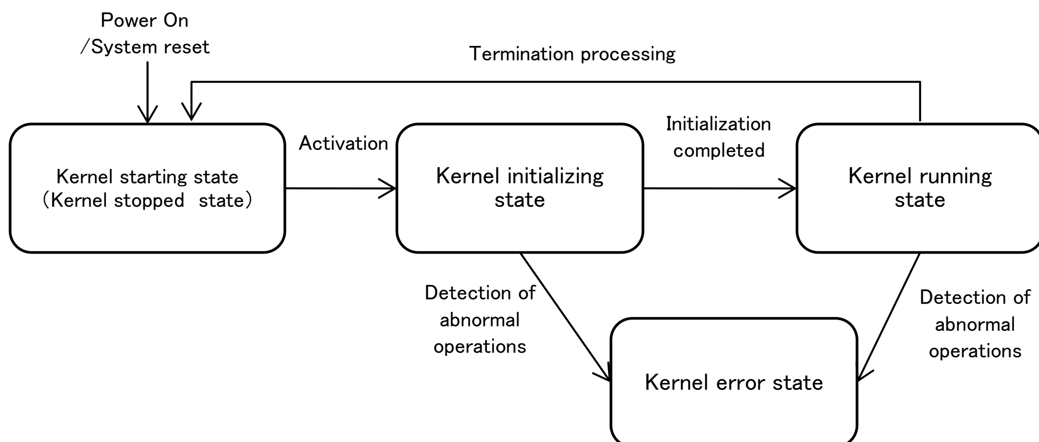

Figure 2-3 State transition of TRON Safe Kernel (Summary)

(3)Kernel running state and (4) Kernel error state have sub-state.

(3)Kernel running state is divided into Task portion running state, Time event handler portion running state, Quasi-task portion running state, Task-independent portion running state and Kernel portion running state according to the executing unit

executed then. Task-independent portion running state is divided into interrupt execution state and failure diagnosis execution state. System state and execution unit are shown in "Table 2-4 System state and Execution unit".

Table 2-4 System state and Execution unit

System state classification	System state	Execution unit
Task portion running state	Task portion running state	Task
Quasi-task portion running state	Quasi-task portion running state	Extended SVC handler
Time event handler portion running state	Time event handler portion running state	Time event handler
Task-independent portion running state	Interrupt execution state	Interrupt handler
	Failure diagnosis execution state	Failure diagnosis handler
Kernel portion running state	Kernel portion running state	Kernel (System call etc.)

(3-1) Task portion running state
A task is running. A running task is recognized as "invoking task". In this state, the running task can invoke the API that causes a wait. In this state, a dispatching can occur whenever a dispatching is not disabled.

(3-2) Quasi-task portion running state
An extended SVC handler called by task is running. An extended handler is executed in the context of the execution unit which invokes the handler. Therefore if a task invokes an extended SVC handler, the handler is recognized as "invoking task" and basically the execution state of the extended SVC handler is the same as the task that invokes the handler. Therefore, a dispatching can occur and an extended SVC handler can invoke the API that causes a wait.

(3-3) Time event handler portion running state
A time event handler is running. Since the running task does not exist, time event handlers cannot invoke the API that specifies the invoking task. And time event handlers cannot invoke the API that causes a wait. Dispatching can always occur.

(3-4) Task-independent portion running state
An interrupt handler or a failure diagnosis handler is running. Since the running task does not exist, these handlers cannot invoke the API that specifies the invoking task. And these handlers cannot invoke the API that causes a wait. In this state, dispatching does not occur basically. Dispatching is pending until this state ends and the state transits to the state where dispatching is possible. This is called "delayed dispatching"

(3-5) Kernel portion running state
TRON Safe Kernel is running. In this state system call and low level interrupt processing are executed. From the standpoint of other programs, this state is executed indivisibly. Other programs cannot recognize this state.

(4)Kernel error state is divided into Abnormal exception state and Safety state.

(4-1) Abnormal exception state

Abnormal exception occurs and an abnormal exception handler is running. In this state the limited API is available basically. If an abnormality exception occurs in Kernel initializing state or Kernel running state, TRON Safe Kernel transits to this state. After executing the necessary processing for an abnormal exception at the abnormal exception handler, TRON Safe Kernel transits to Safety state. However if the abnormal exception handler decides that there is no problem for system execution, TRON Safe Kernel can transit to the previous state.

(4-2) Safety state

All programs including TRON Safe Kernel is stopping. All the interrupt is prohibited and software cannot work at all. It is possible to be transited from abnormal exception state only.

The state transition of kernel state of TRON Safe Kernel including sub-state above mentioned is shown in "Figure 2-4 Kernel state transition of TRON Safe Kernel".

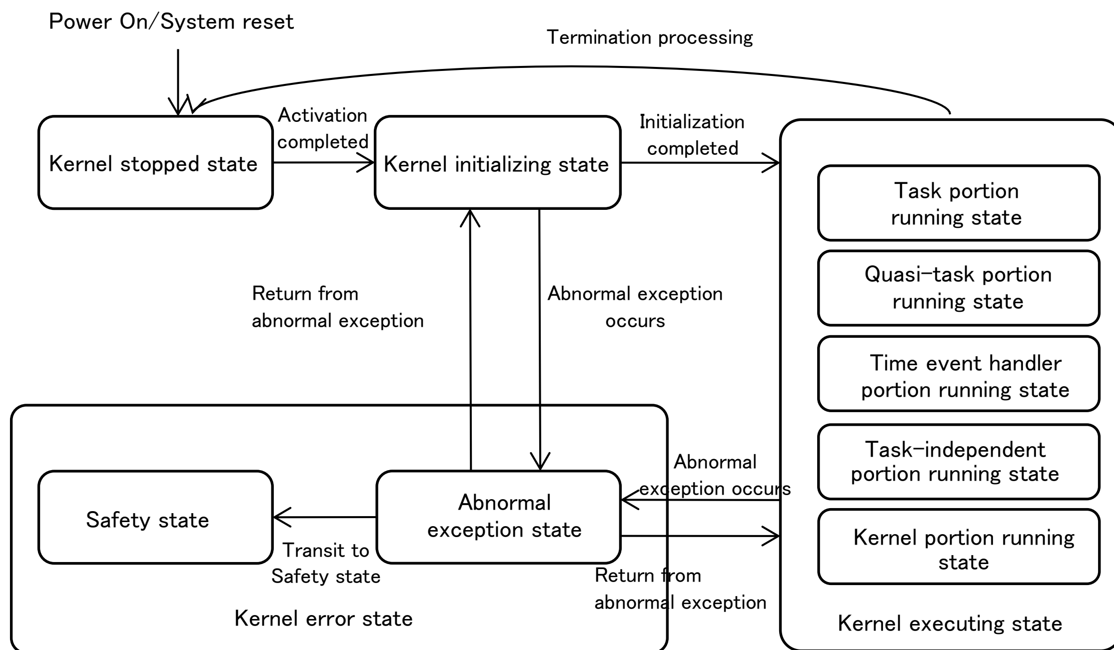

Figure 2-4 Kernel state transition of TRON Safe Kernel

3. TRON Safe Kernel/OS Functions

3.1 Task Management Functions

3.1.1 Overview of Task Management Functions

Task is the executing unit of program operating regularly and is the kernel object.

Function for task management has an operation for task and a control of task dependent synchronization. Task dependent synchronization is realized by operating task state directly.

Each task has information shown Table 3-1 Information for Task Management.. It is possible to refer to the information and change a part of information by API.

Table 3-1 Information for Task Management

Name	Description	Change dynamically *1
Task ID	ID number recognizing a task	NOT possible
Belonged domain	Domain for the task	NOT possible
Protection level	Protection level for task	NOT possible
Task attributes	Attributes for task	NOT possible
Task state	State of task	Possible *2
Task start-up address	Start-up address of executing program for task	NOT possible
Maximum consecutive execution time	Upper limit of the time executing a task consecutively	NOT possible
User stack information	Information for the stack used by task normally	NOT possible
System stack information	Information for the stack used by system on task executing	NOT possible
Start-up priority	Priority for the task set at starting	NOT possible
Base priority	Priority for setting current priority of the task	Possible
Current priority	Current priority of the task	Possible *2
Number of wakeup requests	Number of queues for task wakeup request	Possible *2
Number of suspended requests	Number of nests for suspended request	Possible *2
Reason for disabling of wait	Reason for disabling of wait	Possible *2
Object name	Name for recognizing an object (Character string)	NOT possible

*1 Whether or not it is possible for the information to change by API dynamically after creating

*2 This information is changed as a result of API operations. It is NOT possible to set a value directly by API.

Task ID is assigned by TRON Safe Kernel automatically at creating the task.

Belonged domain means the domain for the task, and is the domain for the program that created the task.

Protection level is set the value defined on the domain for the task.

Task attributes means the task attributes value set at creating the task. Refer to '3.1.2 Task ' for details.

Task state means current task state. Refer to '3.1.3 Task ' for details.

Task start-up address means the address of the program executed at starting the task. Refer to '3.1.4 Main Function for ' for

details.

Maximum consecutive execution time is set as the upper limit of the time possible to execute the task consecutively.

User stack information means information for user stack for the task. System stack information means information for system stack for the task. Refer to '3.1.5 Task ' for details.

Start-up priority, Base priority and Current Priority are used for scheduling for the task. Refer to '3.1.6 Task ' for details.

Number of wakeup requests means the number of wakeup requests queued currently by the task. Refer to '8.2.1.11 ts_wup_tsk/tn_wup_tsk – Wakeup '.

Number of suspended requests means the number of nests for suspended request for the task. Refer to '8.2.1.14 ts_sus_tsk/tn_sus_tsk – Suspend '.

Reason for disabling of wait means the reason for prohibiting waiting for the task. Refer to '8.2.1.18 ts_dis_wai/tn_dis_wai – Disable Task Wait'.

Object name means the name for the task set at creating.

3.1.2 Task Attributes

Task attributes are set at creating the task. Task attributes are shown in "Table 3-2 Task attributes".

Table 3-2 Task attributes

Name	Attributes	Description
TA_ONAME	Set an object name	Set an object name
TA_COP*	Set for using a co-processor	Use a co-processor on a task '*' means an ID number for recognizing a co-processor (implementation-dependent)
TA_FPU	Set for using a FPU	Use a co-processor for floating point operation on a task

3.1.3 Task State

Task states are categorized as follows.

(1) Running state (RUNNING)

Running state means the state executing the task. On executing a program with higher priority other than a task, the task executed previously is on running state.

(2) Ready state (READY)

Ready state means the state unable to execute the task because of a task with higher priority is executing

(3) Waiting state (broad meaning)

Waiting state (broad meaning) means the state unable to execute a task because of not meeting executing conditions for the task, that is, the state waiting for meeting executing conditions for the task. Information for program execution such as program counter, register, etc. is stored on waiting state (broad meaning). If a task transited from waiting state (broad meaning) to on running state, previous information for program execution (having before transited) is resumed.

(3-1) Waiting state (WAITING)

Waiting state means the suspended state of a task execution until any condition is met.

(3-2) Suspended state (SUSPENDED)

Suspended state means the state suspended a task execution forcibly by other task.

(3-3) Waiting-Suspended state (WAITING-SUSPENDED)

Waiting-Suspended state means the overlapping state of waiting state and suspended state. If the task is transited to waiting state and then to on suspended state, its task is on waiting-suspended state.

(4) Dormant state (DORMANT)

Dormant state means the state having no task, not started or already exited.
On dormant state, information for resuming a program execution such as program counter, register, etc. is not stored. The task on dormant state is restarted to execute at the task start-up address.

Task starting means the operation transited the task from a dormant state to on ready state. The state other than the dormant one is called 'Started state' collectively.

The Running state and Ready state are called 'Executable state' collectively.
In the case of the task on ready state having higher priority than that of the task on running state currently, the task transited to ready state is immediately dispatched and transited to running state in some cases. This operation is called 'Previous task on running state is preempted to new one on running state'. The task is also transited to running state immediately after met condition of the task priority order, even if the expression 'Transited to ready state' is described in the function explanation for system call.

Exiting a task means the operation transiting the starting task to dormant state.

State transition of the task is shown in "Figure 3-1 State Transition of Task".

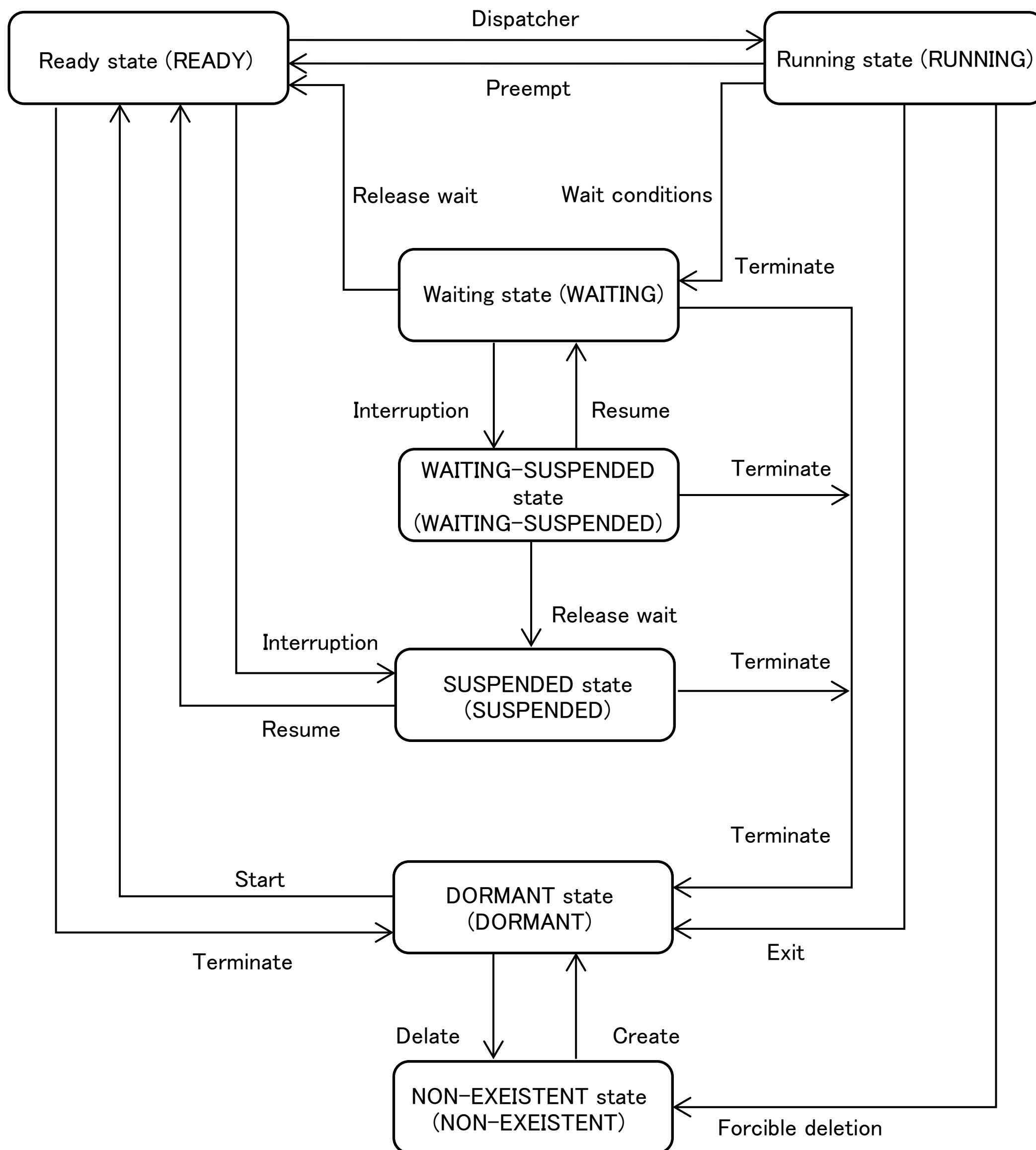

Figure 3-1 State Transition of Task

3.1.4 Main Function for Task

Main function for task means the program executed at starting the task.

Main function for task is the function in C language described below. The function name is arbitrary.

```
void task main (INT stacd)
```

Starting code 'stacd' set by user at starting is passed to the main function for task as argument.

However "0" is definitely passed to the initial task of domain.

3.1.5 Task Stack

Each task has two (2) kinds of stacks, one (1) user stack and one (1) system stack

The user stack is used on executing a program for task.

The system stack is used for the operations for the API such as issuing, etc. on executing a program for task. Conditions for using a system stack are implementation-defined.

The stack size is set statically and the memory area for the stack is assigned statically.

3.1.6 Task Priority

The task has a priority as follows.

(1) Start-up priority

The start-up priority is set at creating a task. When a task starts, all priorities for the task are set to this start-up priority. Also, when exiting and restarting a task, all priorities for the task are set to this start-up priority.

(2) Base priority

The base priority means the basic priority for the task. The base priority is initialized to the start-up priority at creating the task. It is then possible to change by API until the task exiting.

However, refer to section 8.2.1.7 ts_chg_pri/tn_chg_pri – Change Task Priority about the operation using 'ts_chg_pri/tn_chg_pri'

(3) Current priority

The current priority means the priority for task currently. Task scheduling is operated on the basis of this value. 'Task priority', simply called, means the current priority.

The current priority is equivalent to the base priority normally. If locking a Mutex, the current priority is different from the base priority in some cases. Refer to '3.3.3 Mutex' for the current priority in the case of locking a Mutex.

3.1.7 Task Scheduling Rule

The task scheduling on TRON Safe Kernel applies a preemptive priority based scheduling method on the basis of the task priority.

TRON Safe Kernel sets a priority for executable tasks and executes a task with the highest priority based on the rules described below.

- The task with higher current priority is prior to be executed.
- The execution order of the task with same priority is set on the basis of the kind of domain described below.
  System domain (high priority) > Safety domain > Normal domain (low priority)
- If tasks of the same kind have the same priority, the task with smaller domain ID number is prior to be executed (Domain ID number is set statically).
- If tasks on same domain have the same priority, the task transiting to executable state (running state or ready state) earlier is prior to be executed. That is changed by calling API in some cases.

3.1.8 API for Task Management

TRON Safe Kernel has API for task management shown in "Table 3-3 APIs of Task Management Functions" (Refer to Section 8.2.1 'API for Task Management ' about the detail of each API.)

Table 3-3 APIs of Task Management Functions

Category	Name of Safety API	Name of Normal API	Explanation
Create and delete a task	ts_cre_tsk	tn_cre_tsk	Create a task
	ts_del_tsk	tn_del_tsk	Delete a task
Execute a task and exit	ts_sta_tsk	tn_sta_tsk	Start a task
	ts_ext_tsk	tn_ext_tsk	Exit the invoking task
	ts_exd_tsk	tn_exd_tsk	Exit the invoking task and delete
	ts_ter_tsk	tn_ter_tsk	Exit other task forcibly
Refer to information for task and change	ts_chg_pri	tn_chg_pri	Change a task priority
	ts_inf_tsk	tn_inf_tsk	Refer to statistical information for task
	ts_ref_tsk	tn_ref_tsk	Refer to task state
Task depended synchronization	ts_slp_tsk	tn_slp_tsk	Transit the invoking task to waiting state for wakeup
	ts_wup_tsk	tn_wup_tsk	Wakeup other task
	ts_can_wup	tn_can_wup	Cancel a wakeup request for task
	ts_rel_wai	tn_rel_wai	Release other task from waiting state
	ts_sus_tsk	tn_sus_tsk	Transit other task to suspended state
	ts_rsm_tsk	tn_rsm_tsk	Resume a task from suspended state
	ts_frsm_tsk	tn_frsm_tsk	Resume a task from suspended state forcibly
	ts_dly_tsk	tn_dly_tsk	Delay to operate a task
	ts_dis_wai	tn_dis_wai	Disable waiting state of a task
	ts_ena_wai	tn_ena_wai	Release a task from waiting state

3.2 System State Management Functions

3.2.1 Task Scheduling

A task is executed on the basis of the scheduling rules specified in '3.1.7 Task Scheduling s'.

Executing a task according to the scheduling rules in the TRON Safe Kernel is called 'Task scheduling'

Task scheduling is possible by operating the API as follows. The operation is applied to the task on the domain for the task, not to the task on another domain.

(1) Rotate a task priority

Each priority on its respective domain has a queue for executing its task (ready queue for tasks). The task at the head of the queue has the highest priority for executing on the domain.

If a task transitions to an executable state (running state or ready state), it is moved to the back of the queue (with the lowest priority). Tasks are queued in the order that they transition to an executable state.

It is possible for the queue of tasks with the indicated priority on its domain to rotate. The task with the highest priority in the queue for execution is set to the lowest priority. And then, the task with the second highest priority in the queue is set to the highest priority.

(2) Disable Dispatching

It is possible to prohibit a task from dispatching on its domain

Tasks in the running state on the domain when issuing an API continue to be in this state until disable dispatching is released.

.

(3) Get Task Identifier of the task in the running state

It is possible to refer to the ID number of the task in the running state on its domain. The executable task with the highest priority on the domain is in the running state. If dispatch has been disabled for the domain, the Running task for when the domain transitioned to a state of dispatch disabled is the task in the running state, even if its priority has changed.

3.2.2 Reference System Information

It is possible to get system information such as the system state, version number, etc. by API.

(Refer to Section 8.2.2.5 'ts_ref_sys/tn_ref_sys – Reference System Stat' and 8.2.2.6 'ts_ref_ver/tn_ref_ver – Reference Version')

And it is possible to get the version number through the structure 'T_RVER'.

```
typed struct st_rver {
        UH      maker;          /* T-kernel maker code */
        UH      prid;           /* T-kernel identification number */
        UH      spver;          /* Specification version  */
        UH      prver;          /* T-kernel version */
        UH      prno[4];        /* T-kernel products management information  */
} T_RVER;
```

Information on the version number is defined as follows.

The 'maker' is a maker code that implements the kernel of 'TRON Safe Kernel'. The 'prid' is a number that identifies the type of kernel.

The assignment of a concrete value for 'prid' is left to the provider of the TRON Safe Kernel which implements the kernel. However, becausea product is identified by the number only, the provider should consider how to assign the number using a systematic order.

Therefore, it is possible to identify the unique type of kernel by the set of 'maker' and 'prid'.

The original version of the TRON Safe kernel is provided by TRON forum and the 'maker' and 'prid' are as follows.

```
maker = 0x0000
prid = 0x0000
```

The 'spver' indicates the type of OS specification by the upper 4 bits and the version number of the specification that the kernel complies with by the lower 12 bits. For example, the 'spver' of a kernel that complies with Ver 1.00.xx specifications for TRON Safe Kernels is as follows.

```
MAGIC = 0x3 (TRON Safe Kernel)
SpecVer = 0x100 (Ver 1.00)
spver = 0x3100
```

Furthermore, 'spver' corresponding to the draft version (Ver 1.B0.xx) specifications for TRON Safe Kernels is as follows.

```
MAGIC = 0x3 (TRON Safe Kernel)
SpecVer = 0x1B0 (Ver 1.B0)
spver = 0x31B0
```

MAGIC:

Type of OS specification

0x0 TRON common (TAD and so on)

0x1 reserved

0x2 reserved

0x3 TRON Safe Kernel

0x4 AMP T-Kernel

0x5 SMP T-Kernel

0x6 μT-Kernel

0x7 T-Kernel

‘SpecVer’ is the version number of the specification to which the kernel is compliant ‘SpecVer’ is consists of a three-digit BCD code in packed format. In the case of draft specifications, the second digit from the top may be A or B or C.
In this case, the hexadecimal for A or B or C is inserted into the BCD code.
‘prver’ indicates the version number of the kernel’s implementation. The assignment of a concrete value for ‘prver’ is left to the provider of the TRON Safe Kernel which implements the kernel.
‘prno’ is the return parameter for storing information on the kernel product and product number.
The meaning of the concrete value for ‘prno’ is left to the provider of the TRON Safe Kernel which implements the kernel.

3.2.3 API for System Management

The TRON Safe Kernel has API for the system management functions shown in "Table 3-4 API List for System Management Functions"(Refer to Section 8.2.2 'API for System State Management Functions' for details on each API.)

Table 3-4 API List for System Management Functions

Category	Name of Safety API	Name of Normal API	Explanation
Taskscheduling	ts_rot_rdq	tn_rot_rdq	Rotate Ready Queue
	ts_get_tid	tn_get_tid	Get Task Identifier
Disable dispatching	ts_dis_dsp	tn_dis_dsp	Disable Dispatch
	ts_ena_dsp	tn_ena_dsp	Enable Dispatch
Reference state	ts_ref_sys	tn_ref_sys	Reference System State
	ts_ref_ver	tn_ref_ver	Reference Version Information

3.3 Synchronization and Communication Function

3.3.1 Semaphore

A semaphore is a kernel object indicating the availability of a resource and its quantity as a numerical value. A semaphore is used to realize mutual exclusion control and synchronization when using a resource.
A semaphore contains a resource count indicating whether the corresponding resource exists and in what quantity, and a queue of tasks waiting to acquire the resource.

In the case of returning resources, count of semaphore (called 'Semaphore count') is increased by the indicated number.
In the case of getting resources, the semaphore count is decreased by the indicated number. If the semaphore count is not enough, becoming negative value by decreasing, the task is transited to waiting state for getting the returing semaphore resource. A task waiting for semaphore resources is put in the semaphore queue.

To prevent too many resources from being returned to a semaphore, a maximum semaphore count can be set for each semaphore. An error is notified if returned over the limit, (if the semaphore count increases over the limit by indicating number of resources).

The semaphore APIs are shown in "Table 3-5 API List for Functions for Synchronization and Communication (Semaphore) " (Refer to 'Section 8.2.3 API of the Synchronous/Communication Functions (Semaphore) ' about details of each API.

Table 3-5 API List for Functions for Synchronization and Communication (Semaphore)

Category	Name of Safety API	Name of Normal API	Explanation
Create and delete a semaphore	ts_cre_sem	tn_cre_sem	Create a semaphore
	ts_del_sem	tn_del_sem	Delete a semaphore
Operate a semaphore resource	ts_sig_sem	tn_sig_sem	Return a semaphore resource
	ts_wai_sem	tn_wai_sem	Wait for getting a semaphore resource
Refer to semaphore state	ts_ref_sem	tn_ref_sem	Refer to semaphore state

3.3.2 Event Flag

An event flag is a kernel object used for synchronization, consisting of a pattern of bits used as flags to indicate the existence of the corresponding events.
In addition to the bit pattern indicating the existence of corresponding events, an event flag has a queue of tasks waiting for the event flag.

The event notifier sets the specified bits of the event flag by notifying the event.
A task can be made to wait for all or some of the event flag bits to be set. A task waiting for an event flag is put in the queue of that event flag.

The bit pattern of the event flag is set by the UNIT type, and operated by the UNIT type bits.

The event flag APIs are shown in "Table 3-6 API List for Synchronization and Communication Functions" (Refer to '8.2.4 API for Synchronization and Communication Functions (Event Flag)' for details about each API.

Table 3-6 API List for Synchronization and Communication Functions

Category	Name of Safety API	Name of Normal API	Explanation for API
Create and delete an event flag	ts_cre_flg	tn_cre_flg	Create an event flag
	ts_del_flg	tn_del_flg	Delete an event flag
Operate an event flag	ts_set_flg	tn_set_flg	Set an event flag
	ts_clr_flg	tn_clr_flg	Clear an event flag
	ts_wai_flg	tn_wai_flg	Wait for an event flag
Refer to event flag state	ts_ref_flg	tn_ref_flg	Refer to event flag state

3.3.3 Mutex

Mutex is the kernel object for exclusive control among tasks at using sharing resources.

Mutex has the state whether locked or not and a queue for the task waiting for lock. Task locks a Mutex before using a resource. If Mutex already locked by other task, the task is transited to waiting state until unlocking the Mutex, and then queued for the Mutex. Task unlocks a Mutex after finishing using a resource.

Mutex supports Priority inheritance protocol and Priority ceiling protocol in order to protect from a priority inversion without limitation by an exclusive control.

Priority inversion without limitation means the phenomenon a task with high priority waits for finishing executing a task with middle priority 'for the infinite time' because of a task with low priority locking sharing resource with the task with high priority and then the task with low priority preempted by a task with middle priority.

(1) Priority inheritance protocol

Priority inheritance protocol is the method controlling current priority for the task locking a Mutex on the basis of changing the priority for the task waiting for lock.

Current priority for the task locking a Mutex is set the highest one for the Mutex.

(2) Priority ceiling protocol

Priority ceiling protocol is applied by setting 'TA_CEILING' to Mutex attributes.

Priority ceiling protocol is the method controlling a priority for the task on the basis of the upper limit priority for a Mutex. The highest upper limit priority among locked Mutex(s) is set to the current priority of a task (Excluded in case base priority of a task is less than upper limit priority).

Error 'E_ILUSE' is occurred if the task with higher base priority than the upper limit priority for the Mutex with 'TA_CEILING' tries to lock its Mutex. 'ts_chg_pri/tn_chg_pri' notifies the error 'E_ILUSE' if setting the priority over the upper limit priority for the Mutex with 'TA_CEILING' to base priority for the task locking or waiting for locking its Mutex.

In case of using these protocols, current priority for task is changed automatically on the basis of the operation for Mutex in order to protect from the priority inversion without limitation., current priority for task is always adjusted to the maximum value of priority for meeting strictly Priority inheritance protocol and Priority ceiling protocol as follows.

- Base priority for task
- In case of a task locking the Mutex with 'TA_INHERIT', current priority of the task with the highest current priority among the tasks waiting for locking those Mutex(s).
- In case of a task locking the Mutex with 'TA_CEILING', the highest upper limit priority for the Mutex among those Mutex(s).

If operating a Mutex or changed a base priority by 'ts_chg_pri/tn_chg_pri' and then current priority of the task having the Mutex with 'TA_INHERIT' changed, it is necessary for the task locking its Mutex to change current priority in some cases. This is called 'Transitive priority inheritance'. If the task waits for unlocking other Mutex with 'TA_INHERIT', it is necessary for the task locking its Mutex to operate on transitive priority inheritance in some cases.

If changing current priority for task by a Mutex operation, the operation described below is executed.
If the task changed its priority being on executable state, priority order for the task is changed on the basis of the priority, set the lowest priority order among tasks with same priority. If the task changed its priority being queued on the basis of task priority, the order in queue is changed on the basis of changed priority, set the lowest priority order among tasks with same priority.

If Mutex(s) locked by the task exists at exiting its task, those Mutex(s) are unlocked.

API for Mutex are shown in "Table 3-7 API List for Synchronization and Communication Functions (Mutex)" (Refer to '8.2.5 Synchronization and Communication Functions (Mutex)' about the detail of each API.

Table 3-7 API List for Synchronization and Communication Functions (Mutex)

Category	Name of Safety API	Name of Normal API	Explanation for API
Create and Delete a Mutex	ts_cre_mtx	tn_cre_mtx	Create a Mutex
	ts_del_mtx	tn_del_mtx	Delete a Mutex
Operate a Mutex	ts_loc_mtx	tn_loc_mtx	Lock a Mutex
	ts_unl_mtx	tn_unl_mtx	Unlock a Mutex
Refer to Mutex state	ts_ref_mtx	tn_ref_mtx	Refer to Mutex state

3.3.4 Message Buffer

Message buffer is the kernel object operating a synchronization and communication by delivering a message in variable lengths.

Message buffer has a message buffer area for storing a sent message, a queue for the task waiting for sending messages (Queue for sending) and a queue for the task waiting for receiving messages (Queue for receiving).

Massage sender copies a sending message to message buffer. If space of message buffer area is not enough, task waits for sending to the message buffer on the queue for sending until the space of message buffer being sufficient to copy.

Tasks waiting for sending to message buffer are sent each message on the basis of the order in its queue.

For example, if 'Task A' waiting for sending the message in 40 bytes to the buffer and 'Task B' waiting for sending the message in 10 bytes to the same buffer are queued in the order and then the area in 20 bytes is released after other task receiving the message, 'Task B' is unable to send the message until 'Task A' sends the message.

Message receiver picks up one (1) message from a message buffer. If the message buffer has no message, the task is transited to waiting state on the queue for receiving until sending a message. The task on waiting state for receiving from the message buffer is connected to the receive queue of the target message buffer.

Tasks waiting for receiving from message buffer are received each message on the basis of the order in its queue.

If the massage sent to the task on waiting state for receiving from the message buffer, its message is copied directly from buffer for sending to buffer for receiving, not via message buffer area.

Synchronized message function is realized by setting '0' to the size of message buffer area.

Task for sending and task for receiving wait for calling the system call each other, and then the message is delivered at both tasks calling the system call.

API for message buffer is shown in "Table 3-8 API List for Synchronization and Communication Functions (Message buffer)" (Refer to '8.2.6 API for Synchronization and Communication Functions (Message Buffer) ' about the detail of each API.

Table 3-8 API List for Synchronization and Communication Functions (Message buffer)

Category	Name of Safety API	Name of Normal API	Explanation for API
Create a message buffer and Delete	ts_cre_mbf	tn_cre_mbf	Create a message buffer
	ts_del_mbf	tn_del_mbf	Delete a message buffer
Send to message buffer and Receive from	ts_snd_mbf	tn_snd_mbf	Send to message buffer
	ts_rcv_mbf	tn_rcv_mbf	Receive from message buffer
Refer to message buffer state	ts_ref_mbf	tn_ref_mbf	Refer to message buffer state

3.4 Time Management Functions

3.4.1 Overview of Time Management Functions

Time management functions are the function for executing time dependent operations. Time management functions have system time management and time event handler management. Time event handler has cyclic handler and alarm handler.

(1) System time management

TRON Safe Kernel is managed the unique time, called 'System time'. It it only existed in TRON Safe Kernel.
TRON Safe Kernel manages and controls the time for execution unit such as a task, a handler, etc. on the system time basis. The system time is measured by milliseconds. Actual precision and resolution for the system time depending on the hardwear are implementation-defined.
Value of the system time is basically measured from '1985/01/01 00:00:00 (GMT)' by milliseconds. It is possible to define on implementation-defined for requirements of hardware or application. TRON Safe Kernel is operated on the basis of relative time, not effective by the time.

(2) Time event handler management

Time event handler is the execution unit for the program starting on the basis of relativetime. It is a kernel object. Time event handler belongs to the specific domain.
Time event handler has cyclic handler and alarm handler. The cyclic handler is a time event handler starting in the specific interval. The alarm handler is a time event handler starting once at the specific time.

Features of time event handler are indicated as follows.

- Time event handler is executed prior to others on the domain. It is NOT possible to change the priority.
- Time event handler has higher priority order than one of the tasks on the domain.
- Time event handler is unable to be transited to waiting state. It is NOT possible to use the API with on waiting state.

3.4.2 Overview of Time Event Handler

Each time event handler has information shown in the 'Table 3-9 Management Information for Time event handler'. It is possible to refer to the information, and to change some of information by API.

Table 3-9 Management Information for Time event handler

Name	Description	Changed by API *1
Handler ID	ID number recognizing a time event handler	NOT possible
Belonging domain ID	ID number of the domain which a time event handler belongs to	NOT possible
Protection level	Attributes of a time event handler	NOT possible
Handler attributes	Attributes of a time event handler	NOT possible
Handler state	State of a time event handler	possible *2
Starting time	Time for starting the alarm handler (the alarm handler only)	possible
Starting interval	Interval for starting the cyclic handler (the cyclic handler only)	NOT possible
Starting phase of cycle	Phase for starting the cyclic handler (Cyclic handler only)	NOT possible
Hnadler starting code	Passed data at starting a time event handler	NOT possible
Hnadler start-up address	Start-up address for a executed program of a time event handler	NOT possible
Maximum consecutive execution time	Upper limit of the time executing a time event handler consecutively	NOT possible
Use stack information	Information of the stack used for a time event handler normally	NOT possible
System stack information	Information of the stack used for the system on executing a time event handler	NOT possible
Object name	Object name for recognizing an object (by character string)	NOT possible

*1 Whether or not it is possible for the information to change by API dynamically after creating.

*2 Information is changed as the result by operating API. It is NOT possible to set the value by API directly.

Handler ID is assigned by the TRON Safe Kernel automatically at creating a time event handler.

Belonging domain means the domain for a time event handler, equivalent to the domain for the program creating its time event handler.

Protection level is set the protection level defined on the domain for a time event handler.

Handler attributes means the handler attributes value set at creating a time event handler. Refer to '3.4.5 Main Function for Time Event ' in details.

Handler state means the state for current time event handler. Refer to '3.4.4 State of a time event handler' in details.

Handler start-up address means the address of the program executed at starting a time event handler. Refer to '3.4.5 Main Function for Time Event ' in details.

Maximum consecutive execution time is set the upper limit of the time executing a time event handler consecutively.

User stack information means the information for the user stack of a time event handler. System stack information means the information for the system stack of a time event handler.

Object name means the name of a time event handler set at creating.

3.4.3 Attributes for Time Event Handler

Attributes for time event handler is set at creating its time event handler. That is indicated in "Table 3-10 Attributes for time event handler".

Table 3-10 Attributes for time event handler

Name	Attributes	Description	Kind of handler
TA_STA	Start a time event handler	Transit to running state after creating	Cyclic handler only
TA_PHS	Store a starting phase	Store a starting phase of the cyclic handler in stopped	Cyclic handler only
TA_ONAME	Set an object name	Set an object name	All

3.4.4 State of Time Event Handler

State of time event handler is indicated as follows.

(1) Running state

Running state means the state executing a time event handler.

(2) Ready state

Ready state means the state waiting for the time starting a time event handler.

(3) Stopped state

Stopped state means the state not starting a time event handler or the state after finishing executing.

Starting a time event handler means a time event handler is transited from stopped state to on ready state.

Time event handler on ready state is transited to running state if starting time passing and no program with higher priority executed.

Time event handler on running state exits itself after finishing the operations. After the Cyclic handler finishes the operations, it transits to ready status and waits until the next starting time. Alarm handler transits to stopped state after exiting.

State transition of cyclic handler is shown in 'Figure 3-2 State transition of cyclic handler'.

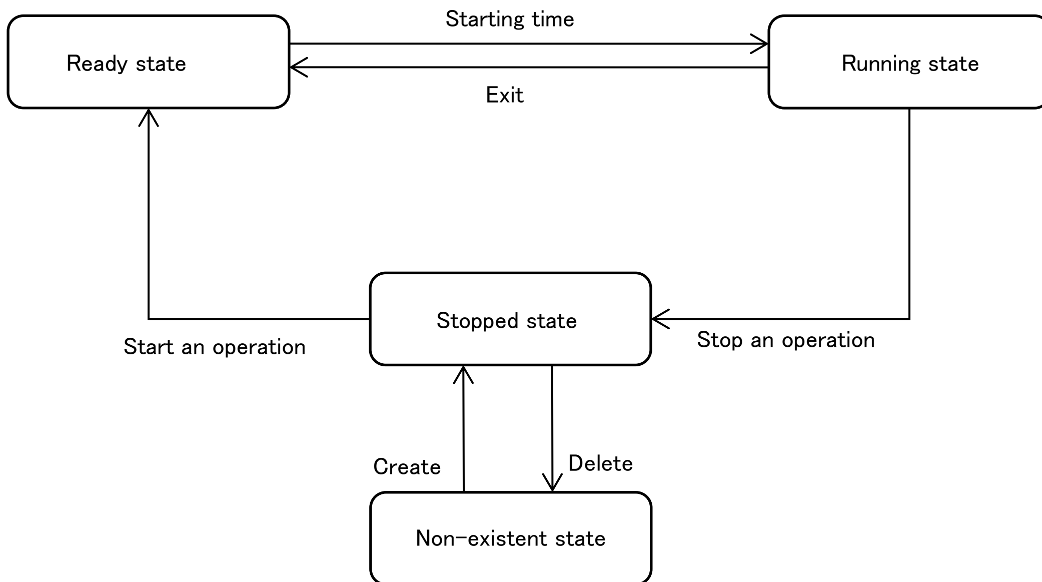

Figure 3-2 State transition of cyclic handler

State transition of alarm handler is shown in ‘Figure 3-3 State transition of alarm handler’.

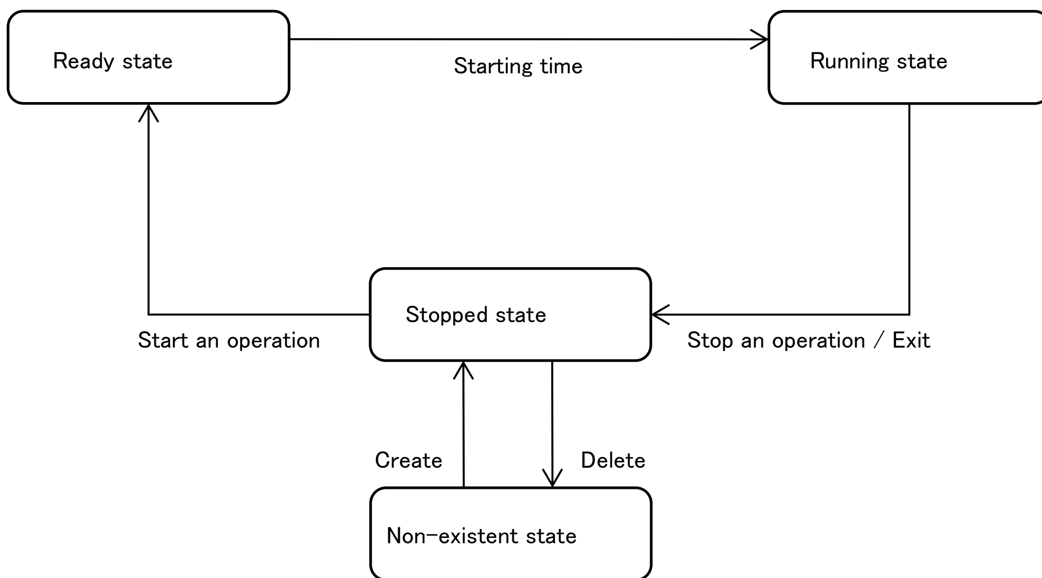

Figure 3-3 State transition of alarm handler

3.4.5 Main Function for Time Event Handler

Main function for a time event handler means the program executing at starting a time event handler.

Main function for a time event handler is a function in C language. Syntax is indicated as follows. It is possible to set the function name. The function name is arbitrary.

```
void hdr_main( INT stacd)
```

Starting code 'stacd' set by user at creating the time event handler is passed as argument for the function.

3.4.6 Scheduling Rule for Time Event Handler

Time event handler is executed on the scheduling rule as follows.

- Time event handler has the highest priority for execution on the domain.
- Time event handler has higher priority order than the tasks with same priority on the domain.
- If time event handlers on the domain has the same starting time, the time event handler transiting to ready state faster has higher priority order.
- Among time event handlers on different kind of domains, priority order for the time event handler or task is set by the rule described below.
  System domain (high priority) > Safety domain > Normal domain (low priority)
- Among time event handlers on different domains of the same kind, priority order for the time event handler or task is set by the rule, that lower ID number of domain has higher priority (ID number of domain is set statically).

Time event handler on running state is not preempted by the time event handler or task on the self domain by the scheduling rule described above, but might be preempted by the time event handler or task on other domain in some cases.

3.4.7 Operation of Cyclic Handler

Starting cycle and starting phase are set to each cyclic handler respectively at creating.

TRON Safe Kernel sets the next starting time for the cyclic handler by the starting cycle and starting phase. TRON Safe Kernel sets the first time by adding the starting phase to the time creating the cyclic handler.

TRON Safe Kernel starts the cyclic handler on ready state with the starting code (stacd) as parameter at the starting time, and sets the next starting time by adding the cycle to the starting time.

If the cyclic handler being on stopped state, TRON Safe Kernel sets the next starting time, not starts the cyclic handler.

If system call for starting the cyclic handler (ts_sta_cyc/tn_sta_cyc) is requested, TRON Safe Kernel transits the cyclic handler to ready state and sets the next starting time if necessary on the handler attributes.

If system call for stopping the cyclic handler (ts_stp_cyc/tn_stp_cyc) is requested, TRON Safe Kernel transits the cyclic handler to stopped state.

Cyclic handler shall not be on running state over the maximum consecutive execution time set at creating. Abnormality exception is occurred if passing the time. Consecutive execution time for the cyclic handler is shorter than cycle time for the cyclic handler.

3.4.8 Alarm Handler

Starting time is set to each alarm handler respectively at creating. TRON Safe Kernel starts the alarm handler with the starting code (stacd) as parameter at the starting time.

Immediately after created, alarm handler has no starting time set and is on stopped state.

If API for starting the alarm handler ('ts_sta_alm/tn_sta_alm') is requested, TRON Safe Kernel sets the starting time by adding the specific time to the issuing time and transits the alarm handler to ready state.

If system call for stopping the alarm handler (ts_stp_alm/tn_stp_alm) is requested, TRON Safe Kernel releases the starting time and transits the cyclic handler to stopped state.

Alarm handler shall not be on running state over the maximum consecutive execution time set at creating. Abnormality exception is occurred if passing the time.

3.4.9 API for Time Management Functions

TRON Safe Kernel time management functions provide APIs shown in 'Table 3-11 API List for Time Management Functions'. Refer to '8.2.7 API for Time Management Functions' about the detail of each API.

Table 3-11 API List for Time Management Functions

Category	Name of Safety API	Name of Normal API	Explanation
System time	ts_set_tim	tn_set_tim	Set the system time
	ts_get_tim	tn_get_tim	Refer to the system time
	ts_get_otm	tn_get_otm	Refer to the system operating time
Cyclic handler	ts_cre_cyc	tn_cre_cyc	Create the cyclic handler
	ts_del_cyc	tn_del_cyc	Delete the cyclic handler
	ts_sta_cyc	tn_sta_cyc	Start the cyclic handler
	ts_stp_cyc	tn_stp_cyc	Stop the cyclic handler
	ts_ref_cyc	tn_ref_cyc	Refer to the cyclic handler status
Alarm handler	ts_cre_alm	tn_cre_alm	Create the alarm handler
	ts_del_alm	tn_del_alm	Delete the alarm handler
	ts_sta_alm	tn_sta_alm	Start the alarm handler
	ts_stp_alm	tn_stp_alm	Stop the alarm handler
	ts_ref_alm	tn_ref_alm	Refer to the alarm handler status

3.5 System Configuration Information Management Functions

3.5.1 Overview of System Configuration Information Management Functions

System configuration information management functions are the function storing and managing information of the system configurations.

A part of system configurations information is defined as the standard system configuration information, and defines the information such as maximum number of tasks, timer interrupt interval, etc. It is possible to add the information defined on applications, subsystems and device drivers.

System configuration information is defined on the Configuration statically. It is NOT possible to change it on TRON Safe Kernel executing.

3.5.2 Format of the System Configuration Information

The format of the system configuration information consists of a name and defined data as a pair.

(1) Name

The name is a string of up to 16 characters. Characters that can be used are shown below.

Characters that can be used (UB) are a to z, A to Z, 0 to 9 and '_' (underscore).

(2) Defined data

Defined data consists of numbers or character strings.

The numbers strings can be retrieved as array of INT type integers.

The character string is a string of up to 128 characters., and characters that can be used are shown below.

Characters that can be used (UB) are any characters other than 0x00 to 0x1F, 0x7F, or 0xFF (in character code).

3.5.3 Standard System Configuration Information

Standard system configuration information is the system configuration information which TRON Safe Kernel must have as a standard. The name of the standard system configuration information has a prefix 'T'.

Standard system configuration information is indicated in 'Table 3-12 List of Standard System Configuration Information'.

Table 3-12 List of Standard System Configuration Information

Name	Type	Number of elements	Explanation
TMaxTskId	N	1	Maximum number of tasks
TMaxSemId	N	1	Maximum number of semaphores
TMaxFlgId	N	1	Maximum number of event flags
TMaxMtxId	N	1	Maximum number of Mutex(s)
TMaxMbfId	N	1	Maximum number of message buffers
TMaxCycId	N	1	Maximum number of cyclic handlers
TMaxAlmId	N	1	Maximum number of alarm handlers

* The information above is defined by each domain ID.

* The system configuration information except 'Table 3-12 List of Standard System Configuration Information', is implementation-defined including whether there is extended information or not and how to implement it.

3.5.4 API for System Information Management Functions

The system configuration information management functions of the TRON Safe Kernel offers API shown in 'Table 3-13 API List for System Configuration Information Management Functions' (Refer to '8.2.8 API for System Configuration Information Management' for the detail of each API.

Table 3-13 API List for System Configuration Information Management Functions

Category	Name of Safety API	Name of Normal API	Explanation
System Configuration Information Management	ts_get_cfn	tn_get_cfn	Acquire the system configuration information (numbers string)
	ts_get_cfs	tn_get_cfs	Acquire the system configuration(character string)

3.6 Domain Management Functions

3.6.1 Overview of Domain Management

Domain is the unit for each program to manage resources on TRON Safe Kernel.

Domain management operates for the domain.

Each domain has information shown in 'Table 3-14 Domain Management Information'. Each information is defined statically at creating the domain. It is possible to refer to the information by safety API.

Table 3-14 Domain Management Information

Name	Description	Changed dynamically *1
Domain ID	ID number recognizing a domain	NOT possible
Protection level	Protection level for program and memory on the domain	NOT possible
Domain attribute	Attribute of domain	NOT possible
Domain state	State of domain	possible *2
Initial task	Task running at the first time on the domain operation	NOT possible
Highest execution priority	Highest priority for task and event handler on the domain	NOT possible
Maximum continuous run time for Task	Upper limit of the time running a task on the domain consecutively	NOT possible
Maximum continuous run time for Time event handler	Upper limit of the time running a time event handler on the domain consecutively	NOT possible
Maximum continuous run time for Domain	Upper limit of the time executing a domain continuously	NOT possible

*1 Whether or not it is possible for the information to change dynamically by API after creating.

*2 Information is changed as the result by API. It is NOT possible to set the value by API directly.

Domain ID is the ID number recognizing the domain defined statically. Refer to '3.6.3 Domain ID and Priority ' in details.

Protection level is the level for the program and the memory on the domain and defined on the basis of the domain kind. Refer to '2.3.2 Protection '.

Domain State means the state of domain. Refer to '3.6.7 State of ' in details.

Initial task is the task executing at creating the domain. Refer to '3.6.4 Create a Domain and Execute'.

Highest execution priority is the highest priority for the task and the time event handler on the domain.

Maximum continuous run time for Task, maximum continuous run time for Time event handler and maximum continuous run time for Domain are the upper limit of continuous run time for the execution unit of the program on the domain. Refer to '3.6.10 Domain Protection on the Time ' in details.

3.6.2 Domain Attribute

Domain attribute is specified at registration of domain. Domain attribute is shown in ‘Table 3-15 Domain Attribute’.

Table 3-15 Domain Attribute

Name	Attribute	Description
TA_START	Attribute to execute at starting	Execute domain at starting TRON Safe Kernel (valid about only Safety domain)

3.6.3 Domain ID and Priority Order

ID number for recognizing a domain is assigned statically to each domain respectively.
ID number for system domain is always '1', for Safety domain consecutive integers in '2' and over, for Normal domain consecutive integers next to the ID number for Safety domain. Relation among ID numbers of the domain is indicated as follows.

System domain ID (1)
Safety domain ID (2) - Safety domain ID (N+1)
Normal domain ID (N+2) - Normal domain ID (N+M+1)

※ N is the maximum number of Safety domain, M is the maximum number of Normal domain.

Domain with lower domain ID number is prior to be executed than others as follows.

- Task and time event handler are executed on the schedule based on the execution priority basically. If tasks and time event handlers having same priority, the task and the time event handler on the domain having higher priority order are prior to be executed.
- Domain with starting execution attributes with lower domain ID number is prior to be executed at TRON Safe Kernel starting.

3.6.4 Create a Domain and Execute

All domains are created statically. Domain is not created and deleted dynamically on executing the program.
TRON Safe Kernel starts (transits to kernel executing state), and system domain is executed, and then Safety domain with starting execution attributes is executed. If multiple Safety domains applied, domain with lower domain ID number is prior to be executed in order. It is NOT possible for Normal domain to be executed at TRON Safe Kernel starting.
Domains not executed on TRON Safe Kernel starting are executed by the Safety API.

'Execute a domain' means the operations for initializing internal information for the domain and executing the initial task. Initial task is the unique task on each domain respectively. It is NOT necessary to issue the API explicitly for creating and executing the initial task because of operating by the kernel.
Task other than initial task, time event handler, and other kernel objects are created by API.

3.6.5 Exit a Domain

Domain exits if meeting the conditions as follows.

- Exit an initial task on the domain.
- Exit the self domain by Safety API or Normal API.
- Exit other domain forcibly by Safety API.

API executes an exit operation at exiting a domain as follows.

- Exit all tasks on executing on the domain forcibly.
- Delete all kernel objects on the domain
- Close all the devices opened by the task belonging to the domain

Exit operation for domain is executed on the context for the initial task on the domain. On the API executing, the domain is transited to dispatch disabled state and the time event handler on the domain is suppressed to execute. Tasks and time event handlers on the the domain are not executed.
Tasks and time event handlers on other domain are executed because of the exit operation executed on the basis of current priority for initial task. Error 'E_NOEXS' is occurred if accessing the kernel objects on the domain under the exit operation.

TRON Safe Kernel exits after the system domain exiting. Exit operation is implementation-defined.
It is possible for an application domain to start again by invoking start domain (ts_sta_dmn) by other domain.

[Supplement]
Other domain cannot terminate system domain. And the initial task of system domain does not complete. Therefore system domain completes only if API of exit of invoking domain is executed by the task in system domain.

3.6.6 Stop a Domain Forcibly

Domain can be stopped forcibly by Safety API in the execution unit on other domain.

It is NOT possible to stop the system domain forcibly. It is defined statically whether it is possible to stop Safety domain forcibly or not depends on the settings when the domain is created.

The task and time event handler on the domain having state stopped forcibly are out of the scheduling of program execution on TRON Safe Kernel and not be executed. State of the task and the time event handler are saved at transiting to forcible stopped state.

Domain is released from forcible stopped state by the Safety API, and then resumed executing.

[Supplement]

To stop a domain forcibly is used in the state that some abnormality (error) occurs in the software in the domain and the execution of the program in the domain is not permitted (domain error state). The application stops the execution of the program in the domain simplyby the cause ause except for error.

3.6.7 State of Domain

Domain has states as follows.

(1) Domain stopped state

Domain stopped state is the state that the tasks and the time event handlers stop before the domain starts. All tasks are on dormant state (DORMANT) and all time event handlers are on stopped state.

Domain is on this state at creating, and started to execute by Safety API and then transits to Domain running state. Only the System domain for application starts from the running state since the domain has already started when TRON Safe Kernel starts.

(2) Domain running state

Domain is in the running state. Tasks and time event handlers on the domain are executed. Initial task on the domain is on the state other than dormant state (DORMANT).

Domain is transited to stopped state via on exit operation state after initial task finishing executing or the domain finishing executing by API.

(3) Domain exit operation state

Domain is in the state of executing an exit operaion. It is transient state from running state to stopped state.

Exit operation is executed on the context for initial task on the domain. Other tasks and other time event handlers is not executed because of the domain transited to dispatch disabled state and the time event handlers suppressed.

(4) Domain forcibly stopped state

Domain is in the state of forcibly stop. All tasks and all time event handlers on the domain are stopped operating.

This state transits from Domain running state by Safety API and returns to Domain running state by Safety API.

State transition of domain is shown in ‘Figure 3-4 State transition of domain’.

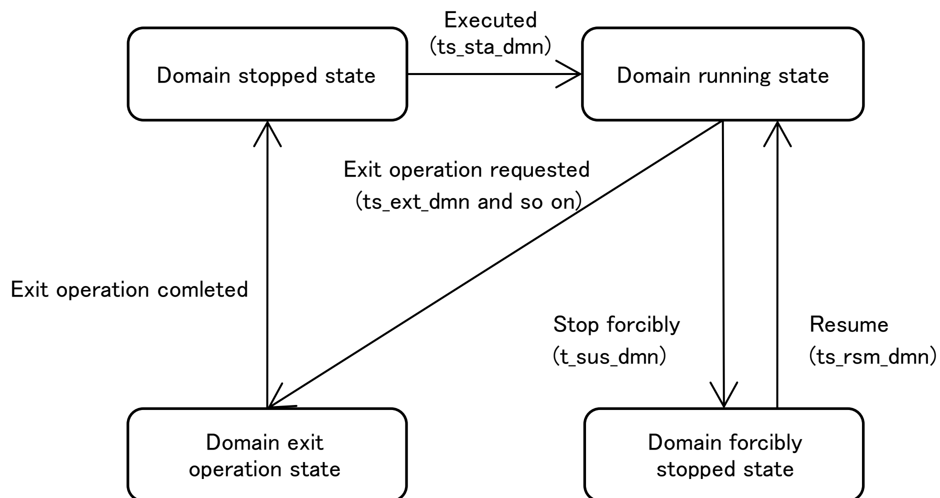

Figure 3-4 State transition of domain

3.6.8 Disable Dispatching

Dispach disabled state is the special state that the dispatch of the tasks on the domain is disabled.

To disable dispatching is executed by API and release of dispatch disabled state is also executed by API. It is possible for the task on the domain to issue its API. The task issuing the API continues to execute during dispatch disabled state.

Operation of the domain in dispatch disabled state is indicated as follows.

- If task having higher priority than the task on running state (the task issuing API to disable dispatching) is executable, dispatching operation is not executed. Its dispatching is delayed until dispatch disabled state ends.
- It is NOT possible for the task on running to transit to the state other than on running state. Error 'E_CTX' is occurred if issuing the API possible to change the state.
- Time event handler is executed. To disable dispatching is valid for the task only in the domain.
- Basically, the task on running state should release dispatch disabled state in timeout time. If dispatch disabled state continues over the timeout time set at API issuing, dispatch disabled state is released and abnormality exception occurs.

[Supplement]

To disable dispatching is only valid on the domain where API is issued. Therefore the task and the time event handler on other domain are executed on the basis of the priority.

From the function safety point of view, to disable dispatching unlimitedly is not permitted. It is requested that the upper limit of priority of task realizing to disable dispatching can be specified and specified upper limit does not exceed the upper limit of priority of the domain which the task belongs to. Therefore at TRON Safe Kernel the range todisable dispatching is limited in the domain which the task who invokes API belongs to. So the upper limit of priority of task at which dispatching is disabled becomes the upper limit of priority of the domain. And it can exclude the impact to the other domain.

3.6.9 Domain Protection on the Space Basis

Memory used by software is protected on the basis of domain on TRON Safe Kernel. This is called 'Domain protection on the space basis'.

Independent memory is assigned statically to each domain respectively. Memory space is not shared among domains.

Memory assigned to domain is protected by memory protection function on processor (MMU, MPU and so on), and then accessing to its memory by software on other domain is prohibited. Abnormality exception is occurred if trying to access its memory illegally.

Application is executed on the execution mode when it is unable to change a memory protection function on processor. It is NOT possible for an application to access a memory area on other domains.

It is possible for system software to change a memory protection function on processor because of its system software executed on the privileged execution mode. It is NOT possible for system software to operate directly a memory area on other domains, except that set directly by API.

3.6.10 Domain Protection on the Time Basis

Software is assigned an execution priority and executed on an execution priority basis on TRON Safe Kernel.

Execution priority and execution time are limited on the domain basis in order to prevent software with higher priority from blocking to execute by software with lower priority. This is called 'Domain protection on the time basis'.

Domain protection on the time basis is indicated as follows.

(1) Lower and Upper limit of the priority

Lowest and Highest execution priority on the domain is the lower and upper limit of execution priority for tasks and time event handlers on the domain. Lowest and Highest execution priority on the domain is set for each domain.

Task has priority in the lowest and highest execution priority or less on the domain.

The time event handler is always executed with the highest execution priority on the domain.

There is no limit of the highest execution priority of System domain.

The highest execution priority of Safety domain is lower (larger value) than the highest execution priority for system domain. Therefore the settable highest execution priority of Safety domain is lower than priority '2' ('2' or more value).

Highest execution priority for Normal domain is lower (larger value) than the highest execution priority for Safety domain.

System domain has no limitation of the highest execution priority. Relation among domains is indicated as follows.

Highest execution priority for System domain (high priority) > Highest execution priority for Safety domain > Highest execution priority for Normal domain (low priority)

(2) Maximum consecutive execution time for task

Maximum consecutive execution time for task is the upper limit of the time executing a task on the domain consecutively.

Task has consecutive execution time in maximum consecutive execution time for task on the domain or less.

(3) Maximum consecutive execution time for time event handler

Maximum consecutive execution time for time event handler is the upper limit of the time executing a time event handler on the domain consecutively. Time event handler has consecutive execution time in maximum consecutive execution time for time event handler on the domain or less.

(4) Consecutive execution time for domain

Consecutive execution time for domain is the time executing a task and a time event handler on the domain. The time is measured consecutively on same domain even if task or time event handler changed.

3.6.11 API for Domain Management Functions

TRON Safe Kernel has API for Domain Management Functions shown in 'Table 3-16 API List for Domain Management Functions' (Refer to '8.2.9 API for Domain Management Functions' for details about of each API.

Table 3-16 API List for Domain Management Functions

Category	Name of Safety API	Name of Normal API	Explanation
Domain execution management	ts_sta_dmn	-	Start domain execution
	ts_ext_dmn	tn_ext_dmn	Finish domain excution
	ts_ter_dmn	-	Finish forcibly other domain execution
	ts_sus_dmn	-	Stop forcivly other domain execution
	ts_rsm_dmn	-	Resume other domain
Kernel object management	ts_get_oid	tn_get_oid	Acquire a kernel object ID
Domain information	ts_ref_dmn	tn_ref_dmn	Get domain state
	ts_inf_dmn	tn_inf_dmn	Refer to domain statistical information

4. TRON Safe Kernel/SM Functions

4.1 Device Management Functions

4.1.1 Device Management Functions Overview

Device management functions manage device drivers running on system domain of TRON Safe Kernel.

A device driver is a program that is implemented independent from TRON Safe Kernel itself to control a hardware device or perform I/O processing with the hardware device.

Device management functions include a function to register the device driver to TRON Safe Kernel, and a function to use the registered device driver from an application.

Device driver has information shown in 'Table 4-1 API List for Device Driver Information'. The information can be referred with API.

Table 4-1 API List for Device Driver Information

Name	Description	Changed dynamically *1
Device ID	ID number that identifies a device driver	NOT possible
Belonging domain	Domain to which the device driver belongs (Fixed to the system domain)	NOT possible
Protection level	Protection level for device driver (Fixed to the protection level 0)	NOT possible
Device driver attributes	Device driver attributes	NOT possible
Number of subunits	Number of device driver subunits	NOT possible
Device driver interface	Interface to device driver for requesting the operation (composing of driver interface functions)	NOT possible
ID for event notification	Message buffer ID for event notification	possible
Object name	Name for identifying an object (string)	NOT possible

*1 Whether or not it is possible for the information to change dynamically by API after creating.

Device ID is assigned automatically by TRON Safe Kernel at registering the device driver.

Belonging domain means the domain for the device driver. ID number of the device driver is fixed to the ID number of system domain because of the device driver always being on the system domain.

Protection level means the level protecting a device driver. Protection level for device driver is fixed to the protection level for system domain because of the device driver always being on the system domain.

Device driver attributes means attributes for device driver. Refer to '4.1.2 Device Driver Attributes' in details.

Number of subunits means number of sub-units for device driver. Refer to '4.1.3 Unit and Subunit' in details.

Interface for device driver means an interface between device management functions on TRON Safe Kernel and device driver. Refer to '4.1.5 Device Driver ' in details.

ID for event notification means ID of the message buffer for sending an event notification. Refer to '4.1.7 Device Event ' in details.

Object name means the name of the device driver set at registering the device driver. Device name is defined on the object name basis. Refer to '4.1.7 Device Event ' in details.

4.1.2 Device Driver Attributes

Device driver attributes are set at registering the device driver. Device driver attributes is shown in 'Table 4-2 Device Driver Attributes'.

Table 4-2 Device Driver Attributes

Name	Attributes	Description
TDA_SDEV	Attributes for Safety device	Device driver that can be handled by Safety API (Available only by Software on Safety domain and System domain)
TDA_NDEV	Attributes for Normal device	Device driver that can be handled by Normal API (Available only by Software on Normal domain)
TDA_OPENREQ	Attributes for multiple open request	Always execute an open operation at multiple opens

The combination of device driver attributes is shown below.

Device driver attributes = (TDA_SDEV || TDA_NDEV) | TDA_OPENREQ

Device driver on TRON Safe Kernel has a driver controllable by Safety API and a drive controllable by Normal API on the basis of device driver attributes because of considering functional safety.

Device driver controllable by Safety API is possible to use by software on Safety domain and System domain.

Device driver controllable by Normal domain is possible to use by software on Normal domain.

It is NOT possible for the device driver to use for both software on Normal domain and software on Safety domain or System domain.

4.1.3 Unit and Subunit

Hardware device has multiple structures in some cases. Whole device is called 'Physical device' and a part of structure in the device is called 'Logical device'.

Same kind of physical devices are classified by 'Unit'. Logical device in the physical device is classified by 'Subunit'.

Device driver supports the unit, that is, the physical device respectively. If physical device has subunits, device driver supports to all subunits on the physical driver.

[Supplement]

Unit and subunit means as follows, example for hard disk.

In case of making a partition in a hard disk, whole hard disk corresponds to physical device and one (1) partition of the hard disk logical device.

In case of two (2) hard disks, one hard disk and another hard disk are classified by the information 'Unit' and one partition and another partition in a hard disk the information 'Subunit'.

4.1.4 Device Name

A device name is given to each device.

The device name is a string up to 8 characters, and composed of a physical device name and a subunit number.

Characters can be used for the physical device name are a to z, A to Z.

The subunit number is a numeric string up to 3 characters, assigned from 0 to 254 in order.

Physical device has physical device name as device name.

Logical device has physical device name and subunit number as device name in order. Total number of characters shall be eight (8) or less. That is called 'Logical device name'.

If there is no subunit, physical device name is equivalent to logical name.

Physical device name is used the name set at registering the device driver.

4.1.5 Device Driver Interface

Device driver interface is the interface for the software between device management functions on TRON Safe Kernel and registered device driver.

Device driver interface is composed of the functions in C language provided by each device driver. Those functions are also called 'Driver interface functions'.

The types of the driver interface functions are shown in 'Table 4-3 List of Driver Interface'. All of these functions are set when registering the device driver. All of the driver processing functions are required for the device driver.

Table 4-3 List of Driver Interface

Function format	Feature
ER initfn(T_INIT_DEV init_dev);	Initialize function (Initialize a device driver)
ER openfn(ID devid, UINT omode);	Open function (Open a device)
ER closefn(ID devid, INT openno, UINT option);	Close function (Close a device)
ER execfn(TS_DEVREQ *devreq, TMO tmout);	Execution function (Request for data input processing to a device / data output processing from a device)
INT waitfn(INT reqno, INT openno, TS_DEVRTN *devrtn, TMO tmout);	Waiting for completion function (Wait for completion of request for data input processing to a device / data output processing from a device)
ER cancelfn (INT reqno);	Request invalid function (Invalidate the request of input and output processing to device)
ER abortfn(INT openno);	Abort function (Abort all the input and output processing for device)
INT eventfn(INT evttyp, void *evtinf, TMO tmout);	Event function (Execute Processing of an event for device)

These driver processing functions are registered by device driver register interface which is possible to call from Safety domain.

Device driver register interface has an interface which sets domain called by hardware access interface of registered device driver.

4.1.6 Input/output Data

API for device driver management allows each device to input and output data.

Data is identified by data number, and is roughly classified into device-specific data and attribute data.

(1) Device-specific data

As device-specific data, data numbers are 0 or more, and defined separately for each device.

(2) Attributes data

Attribute data specifies device driver, device state informations, setting modes, and special functions, etc.

Data number is a negative value. Some of the data numbers are defined commonly, but they are also defined by the device itself.

4.1.7 Device Event Notification

A device driver sends events that occur on each device to the specific message buffer (event notification message buffer) as event notification messages. The event notification message buffer ID is referenced or set as an attribute data of 'TDN_EVENT' for each device.

The system default event notification message buffer is used immediately after device registration. The system default event notification message buffer is created at TRON Safe Kernel startup. Its size and maximum message length are defined by TS_DEVT_MBFSZ_S/TS_DEVT_MBFSZ_N and TS_DEVT_MBFMAX_S/TS_DEVT_MBFMAX_N in the system management information.

The device event notification is classified by the event type.

The major categories of the event types are shown below.

a. Basic event notification (event type: 0x0001 to 0x002F)

Basic event notification from a device

b. System event notification (event type: 0x0030 to 0x007F)

Event notification relating to whole system such as power supply control and so on

c. Event notification with extended information (event type: 0x0080 to 0x00FF)

Event notification from a device with extended information

d. User-defined event notification (event type: 0x0100 to 0xFFFF)

Notification of event that users can arbitrarily define

The message formats used in device event notification are as follows: The content and size of the event notification message vary depending on the event type.

(1) Basic format of device event notification

```
typedef struct st_devevt {
        TDEvtTyp  evttyp;           /* event type */
                                    /* Information specific to each event type is appended here */
} T_DEVEVT;
```

(2) Format of device event notification with device ID

```
typedef struct st_devevt_id {
        TDEvtTyp  evttyp;           /* event type */
        ID        devid;            /* Device ID */
                                    /* Information specific to each event type is appended here */
} T_DEVEVT_ID;
```

(3) Format of device event notification with extended information

```
typedef struct st_devevt_ex {
        TDEvtTyp  evttyp;           /* event type */
        ID        devid;            /* Device ID */
        UB        exdat[16];        /* Extended information */
                                    /* Information specific to each event type is appended here */
} T_DEVEVT_EX;
```

Measures must be taken so that if event notification cannot be sent because the message buffer is full, the lack of notification will not adversely affect operation on the receiving end. One option is to hold the notification until space becomes available in the message buffer, but in that case other device driver processing should not, as a rule, be allowed to fall behind as a result. The specific processing is implementation-defined for each device driver.

Processing on the receiving end should be designed to avoid message buffer overflow as much as possible.

4.1.8 API for Device Driver Management Functions

APIs for Device Driver Management functions are indicated in 'Table 4-4 API List for Device Driver Management Functions' (Refer to 'Section 8.3.2 API for Device Management Functions') in details.

Table 4-4 API List for Device Driver Management Functions

Category	**Name of Safety API**	**Name of Normal API**	**Name of Initial definition API**	Summary description
Register Device	—	—	TS_DEF_DEV	Register Device
Open/Close	ts_opn_dev	tn_opn_dev	—	Open Device
	ts_cls_dev	tn_cls_dev	—	Close Device
Read data asynchronously from a device and Write data asynchronously to a device	ts_rea_dev	tn_rea_dev	—	Request for reading data from a device
	ts_wri_dev	tn_wri_dev	—	Request for writing data to a device
	ts_wai_dev	tn_wai_dev	—	Wait for completion of a request by device
	ts_can_dev	tn_can_dev	—	Cancel a request by device
Read data synchronously from a device and Write data synchronously to a device	ts_srea_dev	tn_srea_dev	—	Read data synchronously from a device
	ts_swri_dev	tn_swri_dev	—	Write data synchronously to a device
Send a device requested event	ts_evt_dev	tn_evt_dev	—	Send a device requested event
Get information for device	ts_get_dev	tn_get_dev	—	Get a device name of device
	ts_ref_dev	tn_ref_dev	—	Refer to device information of device
	ts_oref_dev	tn_oref_dev	—	Refer to device information on the basis of device descriptor

4.2 Interrupt Management Functions

4.2.1 Overview of Interrupt Management Functions

Interrupt management functions are functions to perform operations such as registration of interrupt handlers.

The interrupt handler is the execution unit of a program belonging to the system domain, and is started at interrupting by hardware.

The interrupt management functions can be used only from the system domain software.

And it is not possible to disable interrupts from applications of the safety and the normal domain.

The Interrupts corresponding to TRON Safe Kernel interrupt management functions is applied to the event notification from a peripheral device to a processor and the relating operation.

Event on program execution (such as an address violation, undefined command execution, etc.) and any abnormality detection are operated as an abnormality exception.

The specific interrupt type and cause are to be defined in implementation as depending on hardware.

4.2.2 Overview of Interrupt Handler

Interrupt handler has information shown in 'Table 4-5 Information of Interrupt Handler Management'. The information can be referred with API.

Table 4-5 Information of Interrupt Handler Management

Name	Description	Changed dynamically *1
Interrupt handler number	Number that identifies an interrupt handler	NOT possible
Belonging domain	ID number of the domain to which the interrupt handler belongs (Fixed to the system domain)	NOT possible
Protection level	Protection level for interrupt handler (Fixed to the protection level 0)	NOT possible
Handler attributes	Interrupt handler atttributes	NOT possible
Execution priority	Execution priority for an interrupt handler	NOT possible
Start-up address of an interrupt handler	Start-up address of an execution program for interrupt handler	NOT possible
Maximum consecutive execution time	Upper limit of the time executing an interrupt handler consecutively	NOT possible

*1 Whether or not it is possible for the information to change dynamically by API after creating.

Interrupt handler number is a statically assigned number for identifying an interrupt and in one-to-one correspondence with interrupt factor generated on hardware. Accordingly, it is to be defined in implementation as depending on the hardware.

Belonging domain is the ID number of the domain to which the interrupt handler belongs. Since the interrupt handler always belongs to the system domain, it is fixed to the ID number of the system domain.

Protection level means the level for the interrupt handler. Since the interrupt handler always belongs to the system domain, it is fixed to the protection level of the system domain.

Handler attribute means the interrupt handler attributes. Refer to '4.2.3 Interrupt Handler Attributes' in details.

Execution priority is the priority executing an interrupt handler. Refer to '4.2.4 Execution Priority of an Interrupt ' in details.

Start-up address of an interrupt handler is the start-up address of the program executing at the interrupt handler starting. Refer to '4.2.5 Execution Program for Interrupt ' in details.

Maximum continuous execution time is the upper limit of the time executing an interrupt handler consecutively. Refer to '4.2.7 Maximum Continuous Execution Time for Interrupt ' in details.

4.2.3 Interrupt Handler Attributes

Interrupt handler attributes are specified when interrupt handlers are registered.

Interrupt handler attributes are indicated in ‘Table 4-6 Interrupt Handler Attributes’.

Table 4-6 Interrupt Handler Attributes

Name	Attributes	Description
TIA_MLTINT	Multiple interrupt	Permit a multiple interrupt Permit operating the interrupt with higher priority on an interrupt handler executing

4.2.4 Execution Priority of an Interrupt Handler

Interrupt handler is assigned the specific system priority as execution priority on the basis of interrupt factor. Assignment of the system priority is depend on hardware specification and defined statically. It is NOT possible to change an execution priority for interrupt handler by functions on TRON Safe Kernel. Refer to '2.5.2 Execution ' about the system priority.

If multiple interrupts are occurred at the same time, the interrupt handler with higher priority is executed prior to others.

If interrupt with higher priority is occurred on other interrupt handler executing and attributes for multiple interrupts (TI A_MLTINT) is set, the interrupt handler with higher priority is executed. If attributes for multiple interrupts is not set and the interrupt has same priority or lower, next interrupt is operated after finishing executing current interrupt handler.

Masking an interrupt is depended on hardware specification. Other interrupt handler is executed on current interrupt handler executing without setting attributes for multiple interrupts in some cases. The specification depended on the hardware is implementation-defined.

4.2.5 Execution Program for Interrupt Handler

TRON Safe Kernel calls the registered execution program for an interrupt handler after starting to operate an interrupt. Execution program for interrupt handler is the function in C language in the syntax described below. The function name is arbitrary.

```
void        inthdr( UINT dintno )
```

Interrupt handler number for the interrupt is passed as an argument to the function. The interrupt handler number is the same as the number for executing interrupt handler.

All interrupts are prohibited basically from when the interrupt occurred until the interrupt handler called.

If an interrupt handler executed with attributes for multiple interrupt, the interrupt with higher priority is permitted.

The operation is resumed to previous state (before interrupting) after finishing executing an interrupt handler.

4.2.6 Stack for Interrupt Handler

Interrupt handler uses an exclusive stack for interrupt handler. Size of the exclusive stack for interrupt handler is set statically. Memory area for the stack is assigned statically. Specification for the stack is implementation-defined.

4.2.7 Maximum Continuous Execution Time for Interrupt Handler

Interrupt handler is set the upper limit of the time executing consecutively.

Maximum consecutive execution time means the time from when the interrupt handler called by TRON Safe Kernel until the interrupt hander finishing executing. If multiple interrupt permitted, the time executing an interrupt handler with higher priority is excluded. The time interrupted by the program with higher priority is excluded.

The judge for whether consecutive execution time for interrupt handler is over the setting maximum consecutive execution time for interrupt handler or not is operated at the interrupt handler finishing executing. Interrupt handler is executed after passing the maximum consecutive execution time in some cases.

Abnormality exception is occurred if judged to pass the time over the maximum consecutive execution time at the interrupt handler finishing executing. It is possible for the operation of the time over to use the highest priority timer and to execute by detection using a failure diagnosis function.

4.2.8 API for Interrupt Management Functions

It is possible for the task on the system domain to use API for interrupt management.

API for Interrupt Management Functions are indicated in ‘Table 4-7 API List for Interrupt Management Functions’ (Refer to '8.3.3 API for Interrupt Management Functions' about the detail of each API.).

Table 4-7 API List for Interrupt Management Functions

Category	Name of Safety API	Name of Normal API	Name of Initial definition API	Summary description
Register an interrupt handler	ts_def_int	—	TS_DEF_INT	Register an interrupt handler

4.3 Subsystem Management Functions

4.3.1 Overview of Subsystem Management Functions

Subsystem Management Functions are functions to manage subsystem working on System domain of TRON Safe Kernel.
Subsystem is a program which is implemented as an independent functional module to add or expand functions of system software.
Subsystem Management Functions include functions to register subsystem to TRON Safe Kernel and to use registered subsystem from the other software.

[Supplement]
For application subsystem is a part of system software like TRON Safe Kernel. It is possible to use functions of subsystem by calling extended SVC as to use functions of TRON Safe Kernel by invoking API.

4.3.2 Configuration of Subsystem

Subsystem is composed of following elements.

(1) Extended SVC Handler
The operation to Subsystem is executed by calling extended SVC. Extended SVC handler executes processing of called extended SVC.

(2) Subsystem Interface Functions
Subsystem Interface Functions are functions of interface between Subsystem Management Function of TRON Safe Kernel and Subsystem. They are called from Subsystem Management Functions.

(3) Task and Interrupt Handler
Subsystem can create and use any number of tasks and interrupt handlers. Task and Interrupt handler is not mandatory for subsystem and Subsystem creates, registers and users Tasks and Interrupt handlers as necessary.

Diagram of subsystem is shown in 'Figure 4-1 Diagram of Subsystem'.

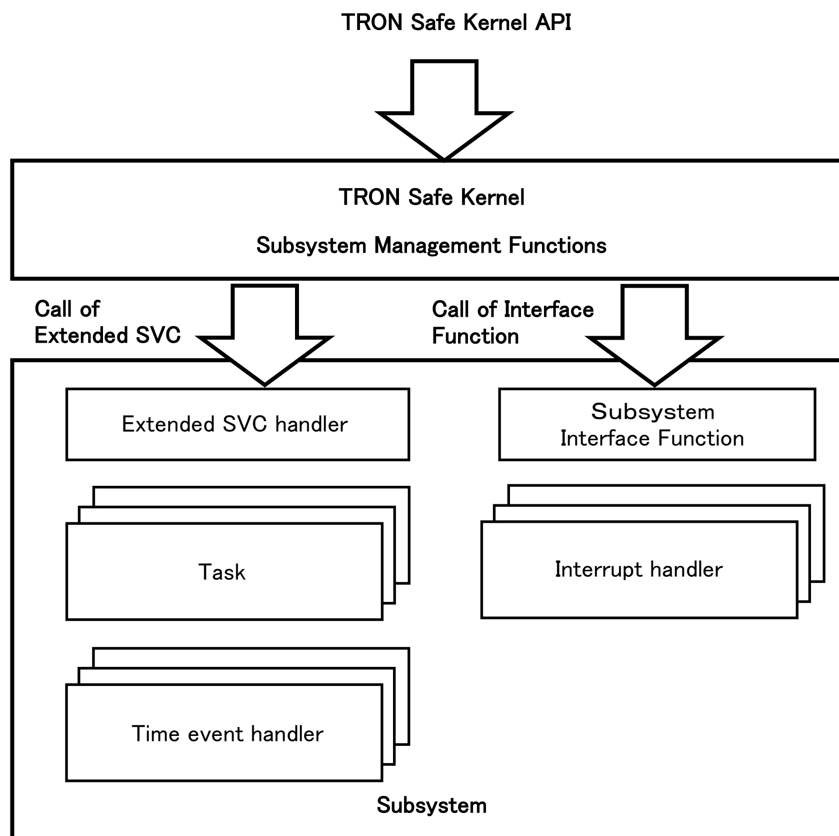

Figure 4-1 Diagram of Subsystem

4.3.3 Information of Subsystem

Each Subsystem has following information in 'Table 4-8 Subsystem Management Information'. Each information is set when Subsystem is registered and it is NOT possible to change the information. A part of information is fixed value and is not necessarily be implemented as actual data.

Table 4-8 Subsystem Management Information

Name	Description	Changed dynamically *1
Subsystem ID	ID Number to identify Subsystem	NOT possible
Belonging domain	Domain to which the subsystem belongs (Fixed to the system domain)	NOT possible
Protection level	Protection level for subsystem (Fixed to the protection level 0)	NOT possible
Subsystem Attributes	Subsystem Attributes	NOT possible
Subsystem priority	Priority for executing interface function of Subsystem	NOT possible
Number of function code	Number of function code provided by Subsystem	NOT possible
Start-up address of extended SVC for Safety API	Start-up address of executing program of extended SVC handler for Safety API	NOT possible

Start-up address of extended SVC for Normal API	Start-up address of executing program of extended SVC handler for Normal API	NOT possible
Subsystem Interface	Interface to request various types of processing to Subsystem	NOT possible

*1 Whether or not it is possible for the information to change dynamically by API after creating

Subsystem ID is a statically specified number to identify Subsystem when Subsystem is registered.

The values of Subsystem ID are from 1 to 255.However the values from 1 to 9 of Subsystem ID are reserved for the system. The values from 10 to 255 are available for Subsystem of user-defined software. However Maximum value of Subsystem ID is implementation-defined and may be less than 255. Actually configurable Maximum Subsystem ID is TS_MAX_SSY. TS_MAX_SSY is statically decided by configuration.

Belonging domain is an ID number of the domain to which the subsystem belongs. Subsystem definitely belongs to System domain, so belonging domain is fixed to ID number of System domain.

Protection level is the one of Subsystem. Subsystem definitely belongs to System domain, so protection level is fixed to the one of System domain.

Subsystem Attributes are the attributes of Subsystem. Refer to ‘4.3.4 Subsystem Attributes’ in detail.

Subsystem priority indicates the priority for executing interface function of Subsystem. Subsystem priority can be set from 1 to TS_LST_SSYPRI and the smaller number becomes higher priority. The range of TS_LST_SSYPRI is from 16 to 255 and it is decided statically by the configuration.

Number of function code indicates the number of function code provided by Subsystem. Function code becomes the argument which is passed to extended SVC handler. Refer to ‘4.3.5 Extended SVC ’ in detail.

Address for starting extended SVC is one for starting program executed when extended SVC is called

Refer to ‘4.3.5 Extended SVC ’ in detail.

Subsystem Interface is the interface between Subsystem management Functions of TRON Safe Kernel and Subsystem. Refer to ‘4.3.6 Subsystem Interface’ in detail.

4.3.4 Subsystem Attributes

Attributes of Subsystem are set when Subsystem is registered.

Subsystem attributes are shown in ‘Table 4-9 Subsystem Attributes’.

Table 4-9 Subsystem Attributes

Name	Attributes	Description
TSA_SSSY	Attribute of Safety Subsystem	Subsystem which provides Safety API (Available by the software of Safety domain and System domain)
TSA_NSSY	Normal of Safety Subsystem	Subsystem which provides Normal API (Possible to use from Normal domain software)

A combination of subsystem attribute is shown below.

Subsystem attribute = TSA_SSSY || TSA_NSSY

4.3.5 Extended SVC Handler

When TRON Safe Kernel accepts the call of extended SVC handler, it calls the execution program of extended handler of the Subsystem. Any function name can be used.

```
ER          svchdr( ID dmnid, FN fncd, T_SVCPARA *pk_svcpara)
```

Argument of the function,'dmnid' is an ID of the domain which invokes ts_cal_svc / tn_cal_svc.
Argument of the function,'fncd' is a function code. A function code indicates a function which extended SVC should execute.
Function codes are consecutive positive values and maximum value is decided by each Subsystem.
Argument of the function,' pk_svcpara' is a parameter which is passed to extended SVC.Its type is defined as below.

```
typedef struct st_svcpara {
        UW          par[TS_MAX_ SVCPARA];
} T_SVCPARA;
```

'TS_MAX_SVCPARA' is a number of implementation-defined which is 4 and over. The content of each member of 'T_SVCPARA' is decided by each Subsystem.
The return value of execution program of extended SVC handler is one of extended SVC.

Extended SVC handler is a part of Subsystem, so it belongs to System domain and is executed by protection level'0'.
Extended SVC handler inherits the program context which calls extended SVC.
The system state executing extended SVC handler called by task becomes Quasi-task portion running state.
At Quasi-task portion running state, task which calls extended SVC is recognized 'invoking task', and the stack uses the system stack of the task.

4.3.6 Subsystem Interface

The interface of the software between Subsystem Management Functions of TRON Safe Kernel and registered Subsystem is called Subsystem Interface. Subsystem Interface is composed of plural functions in C language which each Subsystem provides. These functions are called Subsystem Interface Functions.

Subsystem Interface Functions are called by Subsystem Management Functions of TRON Safe Kernel. Then the order of call is decided by Subsystem priority. In the case of same priority, the order of the priority of the function which has less number of Subsystem ID becomes higher

There are kinds of Subsystem Interface Functions shown in 'Table 4-10 Subsystem Interface List' and all of them are set when Subsystem is registered.

In other words Subsystem should prepare all Subsystem Interface Functions.

Table 4-10 Subsystem Interface List

Function format	Function
ER initfn(T_INIT_SSY init_ssy);	Initialize Function(initialize Subsystem)
void startupfn(ID dmnid);	Startup Function(initialize the internal resource of Subsystem)
void cleanupfn(ID dmnid);	Cleanup Function(release the internal resource of Subsystem)
void breakfn(ID dmnid);	Break Function(break processing of Subsystem)
ER eventfn(ID dmnid, INT evttyp, INT info);	Event Processing Function(execute processing of the request from system to Subsystem)

4.3.7 API for Subsystem Management Functions

API for Subsystem Management Functions are indicated in 'Table 4-11 Subsystem Management Functions API List' (Refer to '8.3.1 API for Subsystem Management Functions' about the detail of each API.).

Table 4-11 Subsystem Management Functions API List

Category	Name of Safety API	Name of Normal API	Name of Initial definition API	Summary description
Register subsystem	—	—	TS_DEF_SSY	Register subsystem
Extended SVC	ts_cal_svc	tn_cal_svc	—	Invoke an extended SVC

4.4 Failure Diagnosis Management Functions

4.4.1 Overview of Failure Diagnosis Management Functions

Failure Diagnosis Management Functions are the functions to manage a failure diagnosis handler which work on System domain of TRON Safe Kernel. A failure diagnosis handler is an execution unit of program executed periodically and at the specified timing to detect the failure of hardware. A failure diagnosis handler belongs to System domain.

Failure Diagnosis Management Functions include the function to register a failure diagnosis handler to TRON Safe Kernel and the function to use registered failure diagnosis handler by other software.

When task or interrupt handler is needed in executing failure diagnosis, it is implemented as Subsystem. Subsystem and a failure diagnosis handler call each other and work together.

4.4.2 Overview of Failure Diagnosis Handler

Each failure diagnosis handler has the information shown in 'Table 4-12 Failure Diagnosis Handler Management Information'. Each information is decided when the handler is registered and cannot be changed. A part of information is fixed value and it is not necessarily implemented as actual data.

Table 4-12 Failure Diagnosis Handler Management Information

Name	Description	Change by API*1
Handler ID	ID Number to identify failure diagnosis handler	NOT possible
Belonging domain ID	Domain ID failure diagnosis handler belongs to (fixed to System domain)	NOT possible
Protection level	Protection level of failure diagnosis handler (fixed to the protection level '0')	NOT possible
Handler attributes	Attributes of failure diagnosis handler	NOT possible
Execution priority of handler	Priority for executing of failure diagnosis handler	NOT possible
Handler status	Status of failure diagnosis handler	NOT possible
Interval of start-up time	Interval to start failure diagnosis handler	NOT possible
Handler start-up address	Handler start-up address of execution program of failure diagnosis handler	NOT possible
Maximum consecutive execution time	Upper limit of time which failure diagnosis handler can execute consecutively	NOT possible

*1 Whether or not it is possible for the information to change by API dynamically after creating

Handler ID is a statically specified number to identify the failure diagnosis handler when the failure diagnosis handler is registered.

Belonging domain is an ID number of the domain the failure diagnosis handler belongs to. The failure diagnosis handler definitely belongs to System domain, so belonging domain is fixed to ID number of System domain.

Protection level is the one of the failure diagnosis handler. The failure diagnosis handler definitely belongs to System domain,

so protection level is fixed to the one of System domain.

Handler attributes are the attributes of the failure diagnosis handler. Refer to '4.4.3 Attributes of Failure Diagnosis ' in detail.

Execution priority of handler is the priority for executing of failure diagnosis handler. The value of task priority can be set to Execution priority.

Handler status is the execution state of the failure diagnosis handler. Refer to '4.4.4 State of Failure Diagnosis ' in detail.

Interval of start-up time is the time interval to start failure diagnosis handler.

Address for starting handler is the address for starting execution program of the failure diagnosis handler. Refer to '4.4.5 Start of Failure Diagnosis ' in detail.

4.4.3 Attributes of Failure Diagnosis Handler

Attributes of failure diagnosis handler are set when the failure diagnosis handler are registered. Handler attributes are shown in 'Table 4-13 Failure Diagnosis Handler Attribute'.

Table 4-13 Failure Diagnosis Handler Attribute

Name	Attributes	Description
TFA_START	Execute when system starts	Execute failure diagnosis handler when system starts
TFA_CYC	Execute periodically	Execute failure diagnosis handler periodically
TFA_API	API operation	Possible to operate of starting or stopping the handler by Safety API

Following combination can be set to the attributes of failure diagnosis handler.

Attributes of failure diagnosis handler = [TFA_START] | [TFA_CYC] | [TFA_API]

4.4.4 State of Failure Diagnosis Handler

States of failure diagnosis handler can be categorized as shown below.

(1) Running state

Running state is the state where the program of failure diagnosis handler is executed.

(2) Waiting state for starting

Waiting state for starting is the state where failure diagnosis handler works and waits for the start-up time.

(3) Stopped state

Stopped state is the state where failure diagnosis handler stops to work.

Failure diagnosis handler is registered in the initialing of TRON Safe Kernel and become Waiting state for starting.
Failure diagnosis handler which states is Waiting state for starting transits to Running state and executes the own program if time reaches start-up time (timing) and the execution unit of the higher priority program is not executed.
Failure diagnosis handler which state is Running state transits to Waiting state for starting after processing and waits for next start-up time.
Failure diagnosis handler which attribute is TFA_API can transit to Stopped state or Waiting state for starting again by using Safety API.
The state transit of failure diagnosis handler is shown in 'Figure 4-2 Failure Diagnosis Handler State '.

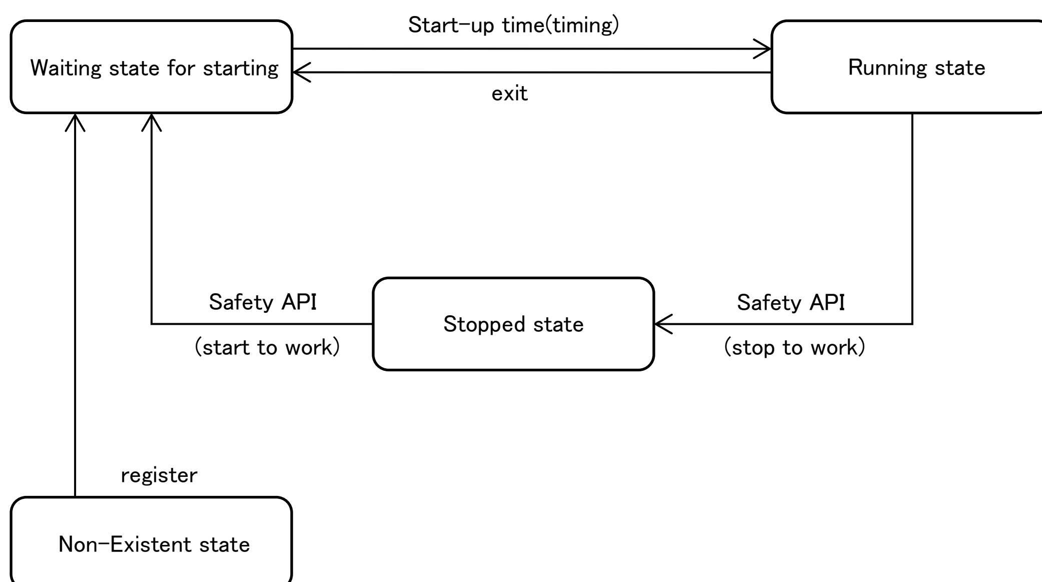

Figure 4-2 Failure Diagnosis Handler State Transition

4.4.5 Start of Failure Diagnosis Handler

Failure diagnosis handler is started according to the attribute at the timing as follows.

- If the attribute of TFA_START is set initialing of TRON Safe Kernel finishes and failure diagnosis handler is started when System domain starts to execute.
- If the attribute of TFA_CYC is set and the state is waiting state for starting, failure diagnosis handler is started periodically according to the interval of start-up time which is set.
- If the attribute of TFA_API is set failure diagnosis handler is started by Safety API.

Started failure diagnosis handler is executed according to the execution priority which is set.
If there are programs whose execution priority is same, the priority is as follows.

failure diagnosis handler (high priority) > time event handler > task (low priority)

If there are failure diagnosis handler whose priority is same, the priority is as follows.

[Supplement]
A failure diagnosis handler can execute in the highest execution priority in System domain. At this time while a failure diagnosis handler is running, it is not dispatched by task and time event handler. However depending on the contents of the failure diagnosis, it may execute in lower priority than task and time event handler. This case is realized by setting the execution priority lower than task and time event handler at the registration of a failure diagnosis hander.

4.4.6 Execution Program for Failure Diagnosis Handler

Execution program for failure diagnosis handler is the function of C language in the format shown below. Any function name can be used.

```
void        fdhdr (void)
```

4.4.7 Time Protection of Failure Diagnosis Handler

If the time of Running state of failure diagnosis handler exceeds the continuous run time, then TRON Safe Kernel generates an abnormal exception. Therefore the execution of failure diagnosis handler is interrupted and the abnormal exception is executed.

If the processing of other execution unit is executed by interrupting while failure diagnosis handler is executed, execution time of the processing is excluded. For example, the execution time of interrupt handler or other failure diagnosis handler are excluded. In this case, continuous run time is held, so when the execution returns to the original failure diagnosis handler, continuous run time is measured consecutively.

Failure diagnosis handler may mask the interrupt while being executed. The period during which the interrupt is masked may not be detected at that point even if it exceeds the continuous execution time. In that case, an abnormal exception occurs when the interrupt mask is released.

4.4.8 Available API by Failure Diagnosis Handler

Available API by an abnormal exception handler is restricted (same as time event handler). And API is available in the case only that a failure diagnosis handler is executed on Kernel running state. If a failure diagnosis handler is executed on the other state and API is issued, ‘E_CTX’ error is returned.

4.4.9 Stack for Failure Diagnosis Handler

A failure diagnosis handler users the dedicated stack of the failure diagnosis handler. The stack size of the dedicated stack of the failure diagnosis handler is statically decided and memory area which the stack uses is assigned statically. The specific specification is implementation-defined.

4.4.10 APIs for Failure Diagnosis Management Functions

APIs for Failure Diagnosis Management Functions are shown below. (Refer to ‘8.3.4 API for Failure Diagnosis Functions’ about the detail of each API)

Table 4-14 Failure Diagnosis Management Functions API List

Category	Name of Safety API	Name of Normal API	Name of Initial definition API	Summary description
Register a handler	—	—	TS_DEF_FDH	Register a failure diagnosis handler
Operate of Failure diagnosis handler	ts_sta_fdh	—	—	Start to work of failure diagnosis handler
	ts_stp_fdh	—	—	Stop to work of failure diagnosis handler
	ts_act_fdh	—	—	Activate failure diagnosis handler

5. Common Rules on TRON Safe Kernel

5.1 Common Data Types on TRON Safe Kernel

5.1.1 General Data Types

General data types on TRON Safe Kernel are difined as follows.

```
typedef signed char          B;      /* Signed 8-bit integer */
typedef signed short         H;      /* Signed 16-bit integer */
typedef signed long          W;      /* Signed 32-bit integer */
typedef signed long long     D;      /* Signed 64-bit integer */
typedef unsigned char        UB;     /* Unsigned 8-bit integer */
typedef unsigned short       UH;     /* Unsigned 16-bit integer */
typedef unsigned long        UW;     /* Unsigned 32-bit integer */
typedef unsigned long long   UD;     /* Unsigned 64-bit integer */

typedef char                 VB;     /* 8-bit data without an intended type */
typedef short                VH;     /* 16-bit data without an intended type */
typedef long                 VW;     /* 32-bit data without an intended type */
typedef long long            VD;     /* 64-bit data without an intended type */
typedef void                 *VP;    /* Pointer to data in without an intended type */

typedef volatile B           _B;     /* 'volatile' declaration */
typedef volatile H           _H;
typedef volatile W           _W;
typedef volatile D           _D;
typedef volatile UB          _UB;
typedef volatile UH          _UH;
typedef volatile UW          _UW;
typedef volatile UD          _UD;

typedef signed int           INT;    /* Signed integer of processor bit width, 32 bits and more */
typedef unsigned int         UINT;   /* Unsigned integer of processor bit width, 32 bits and more */

typedef INT                  ID;     /* General ID */
typedef W                    MSEC;   /* General time (in milliseconds) */

typedef void                 (*FP)();          /* General function address */
typedef INT                  (*FUNCP)();       /* General function address */

typedef UINT                 BOOL;             /* True or false (Boolean values) */
```

VB, VH, VW, and VD differ from B, H, W, and D in that the former mean only the bit width is known, not the contents of the data type, whereas the latter clearly indicate integer type.
Processor bit width must be 32 bits or more. INT and UINT must therefore always have a width of 32 bits or more.
BOOL defines TRUE = 1, but any value other than 0 is also TRUE. For this reason a decision such as bool == TRUE must be avoided. Instead use bool != FALSE.

API parameters that clearly do not take negative values are also in principle signed integer (INT) data type. This is in keeping with the overall TRON rule that integers should be treated as signed numbers as much as possible. As for the timeout (TMO tmout) parameter, its being a signed integer enables the use of TMO_FEVR(= −1) having special meaning. Parameters with unsigned data type are those treated as bit patterns (object attribute, event flag, etc.).

5.1.2 Data Type whose meaning is defined

On TRON Safe Kernel, the following names are used for other data types that appear frequently or have special meaning, in order to make API parameters meaning clear.

```
typedef INT         FN;                 /* Function Codes */
typedef UW          ATR;                /* Attributes */
typedef INT         ER;                 /* Error code */
typedef INT         PRI;                /* Priority */
typedef W           TMO;                /* Timeout specification in milliseconds */
typedef UW          RELTIM;             /* Relative time in milliseconds */

typedef struct systim {                 /* System time in milliseconds */
        W           hi;                 /* High 32 bits */
        UW          lo;                 /* Low 32 bits */
} SYSTIM;
```

5.2 Common Constants for TRON Safe Kernel

5.2.1 General Constants

General constant on TRON Safe Kernel is defined as follows.

```
#define TRUE        (1)     /* True (Boolean values) */
#define FALSE       (0)     /* False (Boolean values) */

#define NULL        (0)     /* Null pointer */
#define TA_NULL     (0)     /* No special attributes indicated */
#define TMO_POL     (0)     /* Polling */
#define TMO_FEVR    (-1)    /* Eternal wait */
```

5.2.2 Symbol Definition

General symbol definition on TRON Safe Kernel is defined as follows.

```
#define LOCAL       static  /* Local symbol definition */
#define EXPORT              /* Global symbol definition */
#define IMPORT      extern  /* Global symbol reference */
```

5.3 Error Code

5.3.1 Overview of Error Code

Error code is the value for notifying an error occurred in operation on TRON Safe Kernel. The code is used for a return value mainly.

An error code consists of the main error code and sub error code. The low 16 bits of the error code are the sub error code, and the remaining high bits are the main error code. Main error codes are classified into error classes based on the necessity of their detection, the circumstances in which they occur and other factors.

Since TRON Safe Kernel does not use a sub error code, these bits are always 0.

5.3.2 List of Error Codes

Error codes and main error codes on TRON Safe Kernel are as follows.

(1) Error class for exit normally

Error code	Main error code (value)	Description
E_OK	0	Normal completion

(2) Internal Error Class

Error code	Main error code (value)	Description
E_SYS	-5	System error Error in the operation on system software occurred
E_NOCOP	-6	Unavailable co-processor eror This error code is returned when the specified co-processor is not installed in the currently running hardware, or abnormal co-processor condition was detected.

(3) Unsupported Error Class

Error code	Main error code (value)	Description
E_NOSPT	-9	Unsupported function error Requested function NOT supported
E_RSFN	-11	Reserved function code number error Requested function code NOT defined This error is occurred when it is attempted to execute an undefined extended SVC handler.
E_RSATR	-12	Reserved attributes error Attributes which cannot be set as the attributes of target Kernel object is set or Conflict attributes set

(4) Parameter Error Class

Error code	Main error code (value)	Description
E_PAR	−17	Parameter error Fail on a value of API parameters This error is occurs in the case as follows. ・In case that the value out of range which can be specified as parameter is set ・In case of pointer value being invalid
E_ID	−18	Invalid ID number error Setting ID number is invalid or NOT used

(5) Call Context Error Class

Error code	Main error code (value)	Description
E_CTX	−25	Context error The API cannot be issued in the current context For example, this error is occurred if issuing the system call for transiting the invoking task to waiting state in executing task independent portion or time event handler or in dispatch disabled state.
E_MACV	−26	Memory access violation error No access authority to the memory area or Memory area NOT existed
E_OACV	−27	Object access violation error No access privilege to the Kernel object
E_ILUSE	−28	System call illegal use Violation of upper limit priority of mutex or multi lock

(6) Resource Constraint Error Class

Error code	Main error code (value)	Description
E_NOMEM	−33	Insufficient memory error Enough memory is NOT allocated
E_LIMIT	−34	System limit exceeded error It is attempted to create more Kernel object(s) than the system allows.

(7) Object State Error Class

Error code	Main error code (value)	Description
E_OBJ	−41	Invalid object state Kernel object state NOT prepared with receiving a request from API
E_NOEXS	−42	Object NOT registered error Setting Kernel object is NOT existed (NOT created) As a common matter about API of domain management function, as all domains are created statically at starting system , and there is no unregistered state, ‘E_NOEXS’ does not occur due to the absence of domain of dmnid. (Refer to 8.2.9 API for Domain Management Functions)
E_QOVR	−43	Queuing or nesting overflow error It is attempted to que and nest more Kernel object(s) than the system allows.

(8) Error class for releasing from waiting

Error code	Main error code (value)	Description
E_RLWAI	−49	Waiting state released error Release the invoking task from waiting state forcibly
E_TMOUT	−50	Timeout error Operation for API is NOT finished in setting time
E_DLT	−51	Waiting object deleted error Kernel object on waiting state is deleted. For example, if semaphore for the task on waiting for getting resource is deleted, the task is released from the waiting state and this error is returned.
E_DISWAI	−52	State of disabling of wait error Waiting factor set by invoking task is disabled

(9) Device Error Class

Error code	Main error code (value)	Description
E_IO	−57	Device error Error occurred on the device driver operating This code is used for the return value of API operating the device driver. Content of the error is returned as a sub-code.
E_BUSY	−65	Busy state error Device drive NOT possible to operate because of on use This code is also notified in case of trying to open the device on an exclusive open.
E_ABORT	−66	Abort error

		Waiting for a completion of device operation aborted

(10) Domain Error Class

Error code	Main error code (value)	Description
E_DOMAIN	−70	Domain error Operation set by API prohibited on the domain for the Kernel object
E_ONAME	−71	Object name error Setting object name already used on the domain
E_DACV	−72	Object access error Operation by the request for Kernel object on other domain prohibited because of a domain protection

5.4 General Specification of TRON Safe Kernel API

5.4.1 API Interface Format

TRON Safe Kernel adopts C as the standard high-level language, and standardizes interfaces for system call execution from C language routines.

The following common rules are established for A interfaces.

- API interface is defined as C language functions.
- A function return code of 0 or a positive value indicates normal completion, while negative values are used for error codes.

Implementation of API is implementation-defined.

5.4.2 API Parameters

Parameter for API is used as an argument for the function defining API in C language.

Some parameters passed to system calls use packet format. Packet is defined by a structure in C language.

Parameter using packet or pointer is added 'CONST' qualifier in order to indicate explicitly no to change the parameter for pointer dependent. 'CONST' is intended 'const' qualifier in C language.

All parameters which is passed to API are checked whether they are within the range of allowed values or not when API calls.

5.4.3 Timeout for API

On TRON Safe Kernel, API that may enter WAITING state has a timeout function. If processing is not completed by the time the specified timeout interval has elapsed, the processing is canceled and API returns error code E_TMOUT.

The calling of the API that times out should in principle result in no change in system state. An exception to this is when the functioning of the system call is such that it cannot return to its original state if processing is canceled. This is indicated in the system call description.

It is possible for timeout interval (TMO type) to set positives only. Standard time for timeout interval (Unit of timeout interval) is the same as the one for system time (by milliseconds). If setting the timeout interval, timeout operation is executed after passing the setting time and over measuring from the time the API being called.

In case the timeout interval is set to 'TMO_POL(= 0)', API is not transited to waiting state even if meeting the condition transiting the API to waiting state. API calling setting 'TMO_POL' to timeout interval is called 'Polling'. API operating a polling has no possibility to be transited to waiting state.

In case the timeout interval is set to 'TMO_FEVR (= -1)', that is setting infinite time to timeout interval, the API waits until meeting the condition for releasing from on waiting state without timeout. It is NOT possible for Safety API to set waiting forever.

The descriptions of individual API as a rule describe the behavior when there is no timeout (in other words, when an eternal wait occurs). Even if the API description states that the API "enters WAITING state" or "is put in WAITING state," if a timeout is set and that time interval elapses before processing is completed, the WAITING state is released and the API returns error code E_TMOUT. In the case ofpolling, the API returns E_TMOUT without entering WAITING state.

5.4.4 Absolute Time and Relative Time

API is set absolute time or relative time as its parameter in some cases.

Absolute time means the system time managed by TRON Safe Kernel.

Absolute time is signed integer in 64 bits SYSTIM type in milliseconds.

SYSTIM type (System time)

```
typedef    struct systim {
           W        hi;                 /* High 32 bits */
           UW       lo;                 /* Low 32 bits */
} SYSTIM;
```

Relative time means the relative value measuring from the time at calling the API and so on.

Relative time is unsigned integer in 32 bits RELTIM type in milliseconds.

RELTIM type (Relative time)

```
typedef    UW       RELTIM;
```

Timeout time for API is defined as in TMO type independently even though its time is a kind of relative time.

TMO type (Timeout time)

```
typedef    W        TMO;
```

Operation with relative time is executed after passing the setting time and over. Operation with timeout time is executed after passing the setting time and over.

TRON Safe Kernel provides a function for setting system time, but even if the system time is changed using this function, there is no change in the real world time (actual time) at which a designated process occurs that was specified using relative time. Similarly, even if the system time changes, the real time that a timeout occurs does not change.

5.4.5 General Specification for Safety API

- General specifications for Safety API are as follows. It is possible for Safety domain to call by Safety software (Software on System domain and the Safety domain) only.
- Name of Safety API has prefix 'ts_'.
- In case of being applicable to the kernel object on normal domain, it is possible for timeout interval to set polling only. If timeout is set except polling, domain error (E_DOMAIN) returns.
- If Safety domain is invoked by Normal domain, an abnormal exception occurs. And if TRON Safe Kernel returns from an abnormal exception domain error (E_DOMAIN) returns from Safety domain invoked.
- If waiting forever is set to timeout interval, domain error (E_DOMAIN) returns.

5.4.6 General Specification for Normal API

General specifications for Normal API are as follows.

- It is possible for Normal API to call by Normal software (Software on the normal domain) only.
- Name of Normal domain has prefix 'tn_'.
- It is NOT possible for normal domain to operate a kernel object on other domain.
  If Normal domain is invoked by Safety domain or System domain, an abnormal exception occurs. And if TRON Safe Kernel returns from an abnormal exception domain error (E_DOMAIN) returns from Normal domain invoked.

5.4.7 Specific Specifications for API

Specific specification of TRON Safe Kernel is defined at 'Section 8 API specification'

6. Safety Functions

6.1 Overview of Safety Functions

TRON Safe Kernel realizes following safety functions to satisfy the request of IEC61508 SIL3 which is Function Safety Standard.

- Detect an abnormal operation

  To prevent the abnormal operation of application of Safety software and System software, TRON Safety Kernel has abnormal exception function. (Refer to ‘6.2 Abnormal Exception Functions’ and ‘6.3 Failure Diagnosis Functions’)

- Divide domains

  To prevent the execution of application of Safety software and System software being inhibited by the execution of application of Normal software, each software is divided by domains. Each domain is spatially and temporally separated. (Refer to ‘6.4 Division of domain’)

- Protection of setting of system software

  Setting of system software is permitted from Safety software. (Refer to ‘6.5 Protection of setting of System software’)

- Transit to safety state

  TRON Safe Kernel provides the function which transits system to safety state when an abnormal operation is detected. (Refer to ‘6.6 Transition to the safety state’)

- Guarantee of response performance

  TRON Safe Kernel provides the function to guarantee the response performance of system interrupt. (Refer to ‘6.7 Guarantee of response performance’)

6.2 Abnormal Exception Functions

6.2.1 Overview of Abnormal Exception Functions

Abnormal Exception Function detects an abnormal operation and is the function to prevent it. Detected abnormal operation is called 'abnormal exception'. The processing corresponding to abnormal exception is called 'abnormal exception handler'. Abnormal Exception Function detects an abnormal exception and is the function to execute the abnormal exception handler corresponding to the abnormal exception.

Abnormal exception is an event in which hardware fault occurs or software execution becomes difficult and is identified by abnormal exception number.

Abnormal exception occurs by the following factors.

(1) Abnormal exception detected by hardware

It is detected by hardware functions. It is composed of address violation, execution of undefined instruction, memory protection violation by MMU or MPU and so on.

(2) Abnormal exception detected by software

It is detected by software functions. There are following kinds of abnormal exceptions.

(2-1) Abnormal exception by TRON Safe Kernel

It is detected while TRON Safe Kernel is working. It is composed of excess of continuous run tim, error during the execution of system call and so on.

(2-2) Abnormal exception by failure diagnosis function

It is detected by failure diagnosis function. (Refer to '6.3 Failure Diagnosis Functions' about failure diagnosis Functions)

(2-3) Abnormal exception defined by user

It is detected by user-defined system software and application of Safety domain. It can be occurred by using Safety API.

The target of an abnormal exception is only hardware failure and the abnormality by which the execution of software is difficult to execute, and if more mild abnormality is detected, abnormal exception does not occur and execution of API ends and error code is notified as a return value of API. Mild abnormality is shown as follows.

- Invoke API which is not permitted
- Invoke API whose argument is out of range permitted

6.2.2 Kinds of Abnormal exception

Abnormal exception is identified by abnormal exception number. Abnormal exception number is categorized as follows.

(1) Standard abnormal exception (abnormal exception number 0x00000000～0x00007FFF)

It may occur in general on TRON Safe Kernel.

(2) Abnormal exception of failure diagnosis (abnormal exception number 0x00008000～0x0000FFFF)

It occurs by failure diagnosis function. It is implementation-defined.

(3) Abnormal exception defined by user (abnormal exception number 0x00010000～0x0001FFFF)

It is occurred by using Safety API by user-defined system software and the application of Safety domain. User can define arbitrarily. It is not defined on the specification of TRON Safe Kernel.

Maximum abnormal exception number is defined by each type of abnormal exception. Maximum number of standard abnormal exception is the maximum number of standard abnormal exception defined by this section. The maximum number of abnormal exception of failure diagnosis and user-defined abnormal exception are defined by configuration.

Standard abnormal exception defined on TRON Safe Kernel is shown in 'Table 6-1 Standard Abnormal Exception List'.

Table 6-1 Standard Abnormal Exception List

Category	Type	Name	Value	Description
Undefined	Undefined abnormal exception	AEXP_UND_AE	0x00000000	Abnormal exception whose abnormal exception handler is not defined occurs
Execute instruction	Undefined instruction	AEXP_UND_CD	0x00000001	Undefined instruction is executed
	Privileged instruction	AEXP_PRI_INS	0x00000002	Privileged instruction is executed by operation mode which cannot be executed
	Divide by zero	AEXP_DIVZ	0x00000003	Division by zero is executed
	Violation of execution area	AEXP_NOEXE	0x00000004	Try to execute memory area which cannot be executed
	Violation of execution Alignment	AEXP_CALI	0x00000005	Alignment of execution code is invalid
FPU	Exception of floating point	AEXP_FPU	0x00000006	Exception about FPU occurs
Memory	Violation of memory	AEXP_MEM	0x00000007	Access the protected

protection	protection			memory area
	Violation of data alignment	AEXP_DALI	0x00000008	Alignment of data access is invalid
Interrupt	Undefined interrupt	AEXP_UND_INT	0x00000009	Interrupt whose interrupt handler is not defined occurs
	Non maskable interrupt	AEXP_NMI	0x0000000A	Non maskable interrupt occurs
Protection on the time basis	Violation of consecutive execution of task	AEXP_TOUT_TSK	0x00000010	Task exceeds consecutive execution time
	Violation of consecutive execution of time event handler	AEXP_TOUT_TEH	0x00000011	Time event handler exceeds consecutive execution time
	Violation of consecutive execution of interrupt handler	AEXP_TOUT_INH	0x00000012	Interrupt handler exceeds consecutive execution time
	Violation of consecutive execution of failure diagnosis handler	AEXP_TOUT_FDH	0x00000013	Failure diagnosis handler exceeds consecutive execution time
	Violation of consecutive execution of domain	AEXP_TOUT_DMN	0x00000014	Domain exceeds consecutive execution time
	Violation of consecutive execution to disable dispatching	AEXP_TOUT_DSP	0x00000015	Time to disable dispatching exceeds timeout time
OS function	Configuration error	AEXP_CFG	0x00000020	Value of configuration information is invalid
	Control block error	AEXP_CB	0x00000021	Data of control block is invalid
	Invoke illegal API	AEXP_ILAPI	0x00000022	Invoke API which function number or number of parameter is illegal
	Invalid exit of task	AEXP_ILEXT	0x00000023	Task exits without invoking API to exit
	Domain or Task exit error	AEXP_EEXIT	0x00000024	Error occurs while API to exit of domain or task is executed
	Message buffer transmission error	AEXP_ERR_MBF	0x00000025	Communication data of message buffer is invalid

	Device driver error	AEXP_DEV	0x00000026	Abnormality occurs at device driver
	Subsystem error	AEXP_SSY	0x00000027	Abnormality occurs at subsystem
	Initial definition API error	AEXP_INIDEF_API	0x00000028	Error occurs at Initial definition API
	Domain start error	AEXP_DMN	0x00000029	Error occurs at domain start

And when an abnormal exception occurs, 3 parameters , that is abnormal exception number (axepno), kernel state (tskstat) when abnormal exception occurs, information (aexppar) defined for each abnormal exception, are passed as argument parameters of abnormal exception handler (Refer to 6.2.6Execution program for Abnormal exception handler). The structure and definition value shown below about information (aexppar) defined for each abnormal exception are used for each abnormal exception number (axepno). Partial content is implementation-defined.

(1) Undefined abnormal exception information

```
typedef struct st_aexp_und_ae {
    UINT      aexpno;              /* abnormal exception number */
} T_AEXP_UND_AE;
```

(2) Undefined instruction information

```
typedef struct st_aexp_und_cd {
    /* implementation-defined */
} T_AEXP_UND_CD;
```

(3) Privileged instruction information

```
typedef struct st_aexp_pri_ins {
    /* implementation-defined */
} T_AEXP_PRI_INS;
```

(4) Divide by zero information

```
typedef struct st_aexp_divz {
    /* implementation-defined */
```

```
} T_AEXP_DIVZ;
```

(5) Violation of execution area information

```
typedef struct st_aexp_noexe {
    /* implementation-defined */
} T_AEXP_NOEXE;
```

(6) Violation of execution Alignment information

```
typedef struct st_aexp_noexe {
    /*   */
} T_AEXP_CALI;
```

(7) Exception of floating point information

```
typedef struct st_aexp_fpu {
    /* implementation-defined */
} T_AEXP_FPU;
```

(8) Violation of memory protection information

```
typedef struct st_aexp_mem {
    /* implementation-defined */
} T_AEXP_MEM;
```

(9) Violation of data alignment information

```
typedef struct st_aexp_dali {
    /* implementation-defined */
} T_AEXP_DALI;
```

(10) Undefined interrupt information

```
typedef struct st_aexp_und_int {
```

```
        UINT    intno;                          /* interrupt number */
    } T_AEXP_UND_INT;
```

(11) Non maskable interrupt information

```
    typedef struct st_aexp_nmi {
        /* implementation-defined */
    } T_AEXP_NMI;
```

(12) Information of violation of consecutive execution of task

```
    typedef struct st_aexp_tout_tsk {
        ID    dmnid;                        /* Domain ID where abnormal exception occurs */
        ID    tskid;                        /* Task ID where abnormal exception occurs */
    } T_AEXP_TOUT_TSK;
```

(13) Information of violation of consecutive execution of time event handler

```
    typedef struct st_aexp_tout_teh {
        ID    dmnid;                        /* Domain ID where abnormal exception occurs */
        ID    tehid;                        /* Time event handler ID where abnormal exception occurs */
    } T_AEXP_TOUT_TEH;
```

(14) Information of violation of consecutive execution of interrupt handler

```
    typedef struct st_aexp_tout_inh {
        ID          dmnid;                  /* Domain ID where abnormal exception occurs (Fixed to System Domain ID)
*/
        ID          inhid;                  /* Interrupt handler ID where abnormal exception occurs */
    } T_AEXP_TOUT_INH;
```

(15) Information of violation of consecutive execution of failure diagnosis handler

```
    typedef struct st_aexp_tout_fdh
    {
```

```
        ID          dmnid;                          /* Domain ID where abnormal exception occurs (Fixed to System Domain ID)
*/
        ID          fdhid;                         /* Failure diagnosis handler ID where abnormal exception occurs */
    } T_AEXP_TOUT_FDH;
```

(16) Information of violation of consecutive execution of domain

```
    typedef struct st_aexp_tout_dmn
    {
        ID          dmnid;                          /* Domain ID where abnormal exception occurs */
    } T_AEXP_TOUT_DMN;
```

(17) Information of violation of consecutive execution of dispatch disabled

```
    typedef struct st_aexp_tout_dsp {
        ID          dmnid;                          /* Domain ID where abnormal exception occurs */
    } T_AEXP_TOUT_DSP;
```

(18) Configuration error information

```
    typedef struct st_aexp_cfg
    {
        T_AEXP_CFG_FACT              factor; /* detail factor */
        T_AEXP_CFG_CLASS              class;   /* class */
        INT                           item;     /* item */
        ER                            ercd;     /* error code */
    } T_AEXP_CFG;

    typedef enum
    {
        AEXP_CFG_OUT_RANGE = 1,               /* out of range */
        AEXP_CFG_DUPLICATTION,                 /* memory area duplication */
        AEXP_CFG_UNDEFINED,                    /* undefined information */
        AEXP_CFG_OVERFLOW,                      /* constitution memory area overflow */
        AEXP_CFG_INITFN,                      /* initialization function error */
        AEXP_CFG_CREATE_ERR                     /* object create error */
    } T_AEXP_CFG_FACT;                        /* detail factor (configuration error) */
```

```
typedef enum
{
    AEXP_CFG_CLASS_SYSTEM = 1,          /* system management information */
    AEXP_CFG_CLASS_MEMORY,              /* memory management information */
    AEXP_CFG_CLASS_DOMAIN,              /* domain management information */
    AEXP_CFG_CLASS_USERSYS              /* user-defined system software */
} T_AEXP_CFG_CLASS;                     /* Abnormal class (configuration error) */

typedef enum {
    AEXP_CFG_ITEM_SYSINF = 1,           /* system information */
    AEXP_CFG_ITEM_OBJINF,               /* object information (common) */
    AEXP_CFG_ITEM_OBJINF_SYS,           /* object information (system) */
    AEXP_CFG_ITEM_OBJINF_S,             /* object information (safety) */
    AEXP_CFG_ITEM_OBJINF_N,             /* object information (normal) */
    AEXP_CFG_ITEM_SSY_MNG,              /* subsystem management */
    AEXP_CFG_ITEM_SDEV_MNG,             /* device management (safety) */
    AEXP_CFG_ITEM_NDEV_MNG,             /* device management (normal) */
    AEXP_CFG_ITEM_INT_MNG,              /* interrupt management */
    AEXP_CFG_ITEM_FD,                   /* failure diagnosis */
    AEXP_CFG_ITEM_AH                    /* abnormal exception */
} T_AEXP_CFG_SYSTEM_ITEM;               /* item (system management information) */

typedef enum {
    AEXP_CFG_ITEM_MEMMAP = 1,           /* memory map */
    AEXP_CFG_ITEM_STACK,                /* stack */
    AEXP_CFG_ITEM_MBF                   /* message buffer */
} T_AEXP_CFG_MEMORY_ITEM;               /* item (memory management information) */

typedef enum {
    AEXP_CFG_ITEM_SYSDMN = 1,           /* System domain */
    AEXP_CFG_ITEM_SAFDMN,               /* Safety domain */
    AEXP_CFG_ITEM_NORDMN                /* Normal domain */
} T_AEXP_CFG_DOMAIN_ITEM;               /* item (domain management information) */

typedef enum {
    AEXP_CFG_ITEM_DDEV = 1,             /* device driver */
    AEXP_CFG_ITEM_INH,                  /* interrupt handler */
    AEXP_CFG_ITEM_SSY,                  /* subsystem */
    AEXP_CFG_ITEM_FDH,                  /* failure diagnosis handler */
```

```
    AEXP_CFG_ITEM_AEH                    /* abnormal exception handler */
} T_AEXP_CFG_USER_ITEM;                 /* item (user-defined system software information) */
```

(19) Control block error information

```
typedef struct st_aexp_cb
{
    T_AEXP_CB_FACT    factor;           /* detail factor */
    VP                error_addr;       /* error occurrence address */
    INT               size;             /* read size */
    UD                read_value;       /* read value */
    UD                clone_value;      /* read value (clone control block) */
} T_AEXP_CB;



typedef enum
{
    AEXP_CB_DATA_ERROR = 1,             /* data error */
    /* Implementation-defined in case of adding below */
} T_AEXP_CB_FACT;
```

(20) Invoke illegal API information

```
typedef struct st_aexp_ilapi
{
    T_AEXP_ILAPI_FACT      factor;      /* detail factor */
    FN                     fncd;        /* function code */
    /* Implementation-defined in case of adding below  */
} T_AEXP_ILAPI;

typedef enum
{
    AEXP_IAPI_FORBIDDEN_STATE = 1,   /* kernel state which is impossible to invoke */
    AEXP_IAPI_UNDEFINED,             /* undefined API */
    AEXP_IAPI_PARAM_MISS,            /* parameter number mismatch */
    AEXP_IAPI_ILLEGAL_CALL_SAPI,     /* illegal call of Safety API */
    AEXP_IAPI_ILLEGAL_CALL_NAPI,     /* illegal call of Normal API */
} T_AEXP_ILAPI_FACT;
```

(21) Invalid exit of task information

```
typedef struct st_aexp_tske
{
    T_AEXP_TSKE_FACT      factor;       /* detail factor */
    ID                    dmnid;        /* domain ID */
    ID                    tskid;        /* task ID */
} T_AEXP_TSKE;

typedef enum
{
    AEXP_TSKE_ILLEGAL_TERM = 1,         /* illegal termination */
    /* Implementation-defined in case of adding below */
} T_AEXP_TSKE_FACT;                     /* detail facor (domain/task exit error) */
```

(22) Domain or Task exit error information

```
typedef struct st_aexp_eexit
{
    T_AEXP_EEXIT_FACT     factor;       /* detail factor */
    ID                    id;           /* domain/task ID */
} T_AEXP_EEXIT;                         /* abnormal exception (domain/task exit error) */

typedef enum
{
    AEXP_EEXIT_DMN_INDEPENDENT = 1, /* domain exit from task independent portion */
    AEXP_EEXIT_TSK_INDEPENDENT,         /* task exit from task independent portion */
    AEXP_EEXIT_TSK_DISDSP,              /* exit from disable dispatch state */
    AEXP_EEXIT_DMN_SYSDMN,              /* exit of System domain */
    AEXP_EEXIT_DMN_FEXT                 /* error occurrence while processing domain abnormal exit */
} T_AEXP_EEXIT_FACT;                    /* detail facor (domain/task exit error) */
```

(23) Message buffer transmission error information

```
typedef struct st_aexp_err_mbf
{
```

```
        ID          mbfid;                     /* message buffer ID */
        VP          data_ptr;                  /* receive data pointer */
    } T_AEXP_ERR_MBF;
```

(24) Device driver error information

```
    typedef struct st_aexp_dev
    {
        T_AEXP_DEV_FACT factor;                /* detail factor */
        ID                id;                  /* device ID / device discripter / request ID */
    } T_AEXP_DEV;                              /* abnormal exception (device driver error) */

    typedef enum
    {
        AEXP_DEV_CLS = 1,                      /* occur at 'ts_cls_dev/tn_cls_dev' */
        AEXP_DEV_CAN,                          /* occur at' ts_can_dev/tn_can_dev' */
        AEXP_DEV_EVT,                          /* occur at 'ts_evt_dev/tn_evt_dev' */
    } T_AEXP_DEV_FACT;                         /* detail factor (device driver error) */
```

(25) Subsystem error information

```
    typedef struct st_aexp_ssy
    {
        ID   ssid;                             /* subsystem ID */
        /* Implementation-defined in case of adding below */
    } T_AEXP_SSY;                              /* abnormal exception (subsystem error) */
```

(26) Initial definition API error information

```
    typedef struct st_aexp_inidef_api
    {
        T_AEXP_INIDEF_API_FACT    factor; /* detail factor */
        UW                        class;  /* class */
        UW                        item;   /* item */
        ID                        id;     /* ID number */
    } T_AEXP_INIDEF_API;                 /* abnormal exception (Initial definition API) */
```

```
typedef enum
{
    AEXP_INIDEF_API_PAR = 1,        /* parameter error */
    AEXP_ INIDEF_API_ID,            /* illegal ID */
    AEXP_ INIDEF_API_LIMIT,         /* value limit over */
} T_AEXP_ INIDEF_API_FACT;          /* detail factor (Initial definition API error) */
```

(27) Domain start error（AEXP_DMN） information

```
typedef struct st_aexp_dmn
{
        T_AEXP_DMN_FACT   factor;   /* detail factor */
        ID                dmnid     /* domain ID */
} T_AEXP_DMN;

typedef enum
{
        AEXP_DMN_START = 1,         /* error occurs at domain start processing */
      /* Implementation-defined in case of adding below */
} T_AEXP_DMN_FACT;
```

6.2.3 Overview of Abnormal exception handler

Abnormal exception handler is the execution unit of program executed when an abnormal exception occurs.

Each abnormal exception handler has information shown in 'Table 6-2 Abnormal exception handler Management Information'. Each information is decided when handler is registered and it is not possible to change it. A part of information is fixed value and is not necessarily implemented as actual data.

Table 6-2 Abnormal exception handler Management Information

Name	Description
Abnormal exception number	Number to identify abnormal exception
Belonging domain	Domain ID abnormal exception handler belongs to (fixed to System domain)
Protection level	Protection level of abnormal exception handler (fixed to the protection level '0')
Handler attribute	Attributes of abnormal exception handler
Handler start-up address	Start-up address of execution program of abnormal exception handler

Abnormal exception number is a statically specified number to identify an abnormal exception. Abnormal exception handler corresponds to abnormal exception one-to-one. Therefore abnormal exception handler can be identified by abnormal exception number.

Belonging domain is an ID number of the domain the abnormal exception handler belongs to. The abnormal exception handler definitely belongs to System domain, so belonging domain is fixed to ID number of System domain.

Protection level is the one of the abnormal exception handler. The abnormal exception handler definitely belongs to System domain, so protection level is fixed to the one of System domain.

Handler attributes are the attributes of the abnormal exception handler. Refer to '6.2.4 Attribute of Abnormal exception handler' in detail.

Address for starting handler is the address for starting execution program of the abnormal exception handler. Refer to '6.2.6 Execution program for Abnormal exception handler' in detail.

6.2.4 Attribute of Abnormal exception handler

Attribute of abnormal exception handler is statically set when abnormal exception handler is registered. Attribute of handler is shown in 'Table 6-3 Abnormal exception handler Attribute'

Table 6-3 Abnormal exception handler Attribute

Name	Attribute	Description
TAA_UPDATE	Possible to change handler	Permit to change abnormal exception handler by Safety API
TAA_RETURN	Possible to return	Possible to return to original state after the exit of abnormal exception handler

6.2.5 Register of Abnormal exception handler

Abnormal exception handler is statically embedded on TRON Safe Kernel and becomes effective by the kernel start processing. Therefore abnormal exception is accepted on Kernel initializing state.

Only abnormal exception handler which attribute is TAA_UPDATE (possible to change handler) can be changed to other abnormal exception handler by Safety API. Abnormal exception handler which attribute is not TAA_UPDATE cannot be changed by API. Therefore the handler which is embedded at first to TRON Safe Kernel continues to be effective.

6.2.6 Execution program for Abnormal exception handler

TRON Safe Kernel calls the execution program of the registered abnormal exception handler when an abnormal exception is accepted. Execution program of abnormal exception handler is the function of C language of the format below. Function name is arbitrary.

```
INT    aehdr( UINT aexpno,   INT tskstat,   VP aexppar )
```

Abnormal exception number of occurred abnormal exception is passed to ‘aexpno’ as a parameter of the function.
Kernel state when an abnormal exception occurs is passed to ‘tskstat’.’aexppar’ is the information defined for each abnormal exception.

TRON Safe Kernel executes an abnormal exception handler at the highest priority when an abnormal exception occurs. Then all the interrupts is prohibited. An abnormal exception handler should not accept interrupts while executing.
An abnormal exception handler whose attribute is ‘TAA_RETURN’ exits the handler by returning ‘AE_RETURN’ (equal to 1) as a return parameter, and it can return to the original state. If an abnormal exception handler returns ‘AE_NORETURN’ (’0’ or value except ‘AE_RETURN) ’ TRON Safe Kernel transits to Safety state and stops the execution of all programs.
If an abnormal exception handler whose attribute is not ‘TAA_RETURN’ exits, TRON Safe Kernel transits to Safety state and stops the execution of all programs.

6.2.7 Available API by Abnormal exception handler

API which is available by an abnormal exception handler is restricted. And API is available in the case only that an abnormal exception handler is executed on Kernel running state. If an abnormal exception handler is executed on the other state and API is issued, 'E_CTX' error is returned.

API which is available by an abnormal exception handler is shown in 'Table 6-4 Available API List from abnormal exception handler'.

Table 6-4 Available API List from abnormal exception handler

Classification	Name	Function
Synchronization control	ts_wup_tsk	Wakeup task
	ts_sig_sem	Signal Semaphore
	ts_set_flg	Set event flag
Get information	ts_ref_sys	Reference system state
	ts_ref_dmn	Reference domain state
	ts_get_tim	Get system time
	ts_get_otm	Get system operating time

6.2.8 Stack for Abnormal exception handler

An abnormal exception handler users the dedicated stack of the abnormal exception handler. The stack size of the dedicated stack of the abnormal exception handler is statically decided and memory area which the stack uses is assigned statically. The specific specification is implementation-defined.

6.2.9 API for Abnormal exception functions

API for abnormal exception functions is shown in 'Table 6-5 Abnormal exception functions API List'. (Refer to '8.3.5 API for Abnormal Exception Functions' about the detail of each API.)

Table 6-5 Abnormal exception functions API List

Category	Name of Safety API	Name of Normal API	Name of Initial definition API	Explanation
Register handler	ts_def_aeh	—	TS_DEF_AEH	Register an abnormal exception handler
Operation of handler	ts_ras_aexp	—	—	Occur abnormal exception

6.3 Failure Diagnosis Functions

TRON Safe Kernel has a function to execute failure diagnosis of hardware and a function to self-diagnose kernel to detect failure.

If a failure is detected, an abnormal exception function is executed as an abnormal exception.

The list of Failure Diagnosis Functions is shown in ‘Table 6-6 Failure Diagnosis Functions List’.

Table 6-6 Failure Diagnosis Functions List

Name of diagnostic function	**Target of Diagnosis**	**References**
Self-diagnosis of kernel	Diagnosis of control block	Section 6.3.1
	Diagnosis of interrupt handler of system timer	
	Detect transmission error of message buffer	
Failure Diagnosis of hardware	Failure diagnosis which depends on specific hardware	Section 6.3.2

6.3.1 Self-diagnosis of kernel

It is a function to detect the failure of TRON Safe Kernel itself. There are following items in Self-diagnosis of kernel.

（1）Diagnosis of Control Block

Control block is the memory area which stores data about the control of TRON Safe Kernel. Control block has a copy of the data and save the copy data by inverting bits of original data. Error is detected by comparing the original data and copy data when the data is read. If error is detected, an abnormal exception occurs.

（2）Diagnosis of system timer interrupt handler

TRON Safe Kernel checks that it can invoke interrupt handler correctly by executing periodically self-diagnosis of interrupt handler which it uses.

Particularly the interrupt handler is the one of system timer. An interrupt handler is executed in a defined interval and Reset of the highest priority timer is executed in the processing.

If this processing does not work normally, the highest priority timer is not reset and interrupt of the highest priority timer occurs.

（3）Detect transmission of message buffer

This function detects error of transmission content at the transmission using message buffer(3.3.4 Message).To detect the error of transmission content by CRC（Cyclic Redundancy Check） is executed to detect the transmission error.

If the error is detected by CRC an abnormal exception occurs.

Whether self-diagnosis function of kernel is valid or not can be set statically to each item. User sets valid or invalid according to the design of function safety of total system.

[Supplement]

Setting of invalid of self-diagnosis of kernel is used in the case that the diagnosis is not necessary on the design of function safety of total system. For example, it is considered that a failure is detected by hardware and self-diagnosis of kernel is not

necessary.

6.3.2 Hardware failure diagnosis

The target of self-diagnosis of kernel of TRON Safe Kernel is only the item which is independent of particular hardware (Refer to 6.3.1 Self-diagnosis of kernel). Therefore, the processing of hardware failure diagnosis which is not the target of self-diagnosis of kernel is realized as a user-defined system software (device driver, subsystem, interrupt handler)

If the execution of processing of hardware failure diagnosis is needed, a failure diagnosis handler of failure diagnosis management function of TRON Safe Kernel is available. A failure diagnosis handler is executed at the specified timing and executes the processing of hardware failure diagnosis realized by a user-defined software (Refer to 4.4 Failure Diagnosis Management Functions in detail).

The content of diagnosis depends on hardware and is implementation-defined.

[Supplement]

If the processing of hardware failure diagnosis is simple, it is possible to execute failure diagnosis by failure diagnosis handler only.

6.4 Division of domain

6.4.1 Spatial separation of the domain

Spatial separation of the domain is realized by allocating independent resource for each domain and protecting the resource from the other domain. The resource allocated to domain is memory area and kernel object which the program belonging to the domain uses.

Spatial separation of the domain is realized as follows.

(1) Allocation of resource

Memory area and kernel object are allocated independently for each domain and not shared.

Allocation of memory area of Safety domain and System domain is executed statically by the configuration at the time of system construction. The resource (memory) which kernel object uses is decided statically by the configuration at the time of system construction.

Allocation of resource of Normal domain is basically equivalent to Safety domain; however a part of memory area such as stack of task is allocated dynamically. However memory area allocated is statically defined by the configuration at the time of system construction.

(2) Protection of resource

The resource allocated for each application domain is protected from the access of the other domain by the hardware mechanisms of memory protection function. (Refer to 3.6.9 Domain Protection on the Space in detail)

Kernel object is managed in Safety domain and Normal domain independently. Particularly the control block of kernel object is secured in the different memory area in Safety domain and Normal domain and the operation to the control block is possible through the internal control block interface of TRON Safe Kernel. Therefore the access to kernel object allocated to Safety domain and operation to the kernel object which is not permitted from Normal application is prevented.

(3) Prevention of failure propagation

It is prevented as follows that the failure of program of Normal domain propagates the program of Safety domain.

The operation to the kernel object of Normal domain from the program of Safety domain can be executed by API of TRON Safe Kernel only. The propagation of the value of API which is not permitted is prevented by checking the parameter of API (defensive programming).

In addition, API which operates kernel object of Normal domain from the program of Safety domain has no state to wait. Therefore wait of program of Safety domain can be released by the failure of Normal domain. (Refer to 2.6.4 Protection among Domains for Kernel in detail)

[Supplement]

Not only the operation of Normal domain from Safety domain but also the operation of kernel object is possible through API only. Therefore defensive programming is valid at all the operation of the kernel object.

6.4.2 Temporal separation of Domain

Temporal separation of domain is to allocate time resource (CPU operating time) for each domain by setting the execution priority and the consecutive execution possible time of the program to the domain and execution unit belonging to the domain. If the execution time exceeds the consecutive execution time which is set, it is detected that the execution of the program of a domain is inhibited by the execution of the program of the other domain by detecting as an abnormal exception.
Temporal separation of domain is realized as follows.

(1) Control of execution priority

Execution priority is allocated to all the execution unit of the program. TRON Safe Kernel performs a preemptive priority-based scheduling system based on the execution priority. Particularly TRON Safe Kernel selects and executes the highest one of the executable task and time event handler by the internal scheduler and dispatcher of TRON Safe Kernel. The scheduler is executed at API execution about task start-up and the period of system timer.
If the priority of an interrupt handler and abnormal exception handler is higher than that of the execution unit of the running program, they are executed by interrupting the execution unit of running program.
The upper and lower limit of the common priority can be specified to the execution unit of all the program of the domain. There is no upper limit of System domain. The maximum value of configurable upper limit and lower limit are specified to Safety domain.
The priority of the execution unit of each program cannot be specified over the upper and lower limit of the priority specified in the belonging domain.

(2) Control of maximum execution time

It is realized by following function that the execution unit of the particular program does not inhibit the execution of the execution unit of the other program whose priority is higher or same by continuing to occupy CPU.

- The maximum continuous run time which is possible to execute consecutively is specified to the execution unit of the program and domain.
- It is checked at the time management of TRON Safe Kernel whether the consecutive execution time of the execution unit of the running program exceeds the maximum continuous run time or not. If the time exceeds the maximum continuous run time, abnormal exception occurs. And the total execution time and consecutive execution time can be getting by API.
- The maximum valid time is specified to time of dispatch disabled. It is checked at the time management of TRON Safe Kernel whether time of dispatch disabled exceeds the maximum valid time or not. If the time exceeds the maximum valid time, abnormal exception occurs.
- Dispatch disabled is valid in the domain only which the task invoking API of dispatch disabled belongs to. Therefore the upper limit of task priority which can be the target of dispatch disabled in the time of dispatch disabled is the upper limit specified to the domain which task which executes dispatch disabled belongs to.
- Time event handler is executed by interrupting the execution unit of the program whose priority is lower after the start-up time specified at the time management of TRON Safe Kernel.
- It is checked whether the processing time of an interrupt handler exceeds the maximum consecutive interrupt execution time at the end of each interrupt processing. If the time exceeds the maximum consecutive interrupt execution time, abnormal exception occurs.

6.5 Protection of setting of System software

The operation to register user-defined system software (subsystem, device driver, interrupt handler and so on) by application is prohibited.

The registration of user-defined software is possible by only the Initial definition API defined in the configuration.

And domain which is possible to invoke (Safety domain or Normal domain) is specified at the registration of subsystem and device driver.

6.6 Transition to the safety state

When an abnormality is detected, TRON Safe Kernel executes corresponding abnormal exception handler. The processing of abnormal exception handler is implemented by user. An abnormal exception handler can specify return to the original state or transition to the safety state of TRON Safe Kernel on completion. The safety state of TRON Safe Kernel is to stop the execution of the execution unit including task, time event handler and interrupt handler.

An abnormality of an abnormal exception handler itself is detected by the highest priority timer. The operation when the highest priority timer detects an abnormality is defined by user. TRON Safe Kernel itself stops all the operations after the highest priority timer detects an abnormality.

[Supplement]

The highest priority timer is a watch dog timer to detect the failure of execution itself of system software. The operation of the highest priority timer is not defined at TRON Safe Kernel.

6.7 Guarantee of response performance

Interrupt response performance of TRON Safe Kernel is guaranteed by following mechanism.

- The maximum time from occurrence of interrupt to start of processing of interrupt handler can be predicted by calculating the maximum delay time of interrupt defined by hardware and processing time of execution instruction of interrupt processing mechanism of this software.
- TRON Safe Kernel checks whether the processing time exceeds the default execution time or not at the completion of interrupt handler. If the time exceeds the default execution time, abnormal exception is issued.
- Application cannot prohibit interrupt.

6.8 Debug functions

A debug function provides the fault injection to system software and the simulation of fault injections to memory and I/O. A debug function is not implemented as a function of TRON Safe Kernel. If necessary it is implemented as user-defined system software (device driver, subsystem, interrupt handler and failure diagnosis handler). Therefore a debug function and the software to realize it is not included in TRON Safe Kernel itself.

7. Configuration of TRON Safe Kernel

7.1 Overview of Configuration

To set various kinds of system of TRON Safe Kernel and to define the software embedded at initial state is called ‘configuration’.

Configuration is described in the configuration definition file. Configuration is specified at the construction of TRON Safe Kernel and cannot be changed at running.

7.1.1 Configuration Information

Configuration information is various kinds of static data about system.

Configuration information is constant. Particularly numeric data defined by macro definition (#define) of C language or const data (const type).Configuration information is shown below.

- System management information (Section 7.2)

  This is the information to set the resource number of system and limited value of TRON Safe Kernel. For example, definition of domain and the maximum number of task correspond to it.

- Memory management information (Section7.3)

  This is the management information about the resource of memory. For example, memory-map and information of stack area correspond to it.

- Domain management information (Section7.4)

  This is the management information about the domain. It is the information to set the number of resource and limited value defined by each domain. For example, the highest priority which is possible to define by domain corresponds to it.

7.1.2 Initial definition API

Initial definition API registrates the software embedded at the initial state of TRON Safe Kernel.

The targets are the following user-defined system software.

- Device driver
- Interrupt handler
- Subsystem
- Failure diagnosis handler
- Abnormal exception handler

Initial definition API is defined in the format of C language function as well as other API of TRON Safe Kernel. However Initial definition API can be described at the stated point only of configuration definition file and it is not available in the program.

Initial definition API is described by following rules.

(1) Name

Name of Initial definition API is composed of upper-case alphabets and “” and begins at “TS”. Name of Initial definition API is the format of “TS_XXX_YYY”. ”XXX” shows function (operation) and ”YYY” shows target of operation.

(2) Parameter

Parameter of Initial definition API is either integer constant or string. Detail is defined at individual Initial definition API.

7.1.3 Execution of Configuration

Configuration information is described in the configuration definition file.

The configuration definition file is for each type of configuration. The list of configuration definition file is shown in 'Table 7-1 Configuration definition file List'.

Table 7-1 Configuration definition file List

Category	File name	Contents
System management information	SYSCONF.CNF	System management information
Memory management information	MEMCONF.CNF	Memory-map management information
	STKCONF.CNF	Stack management information
	MBFCONF.CNF	Message buffer management information
Domain management information	DMNCNF.CNF	Domain management information
Registered information of user-defined software	DEVCONF.CNF	Registered information of device driver
	INTCONF.CNF	Registered information of interrupt handler
	SSYCONF.CNF	Registered information of subsystem
	FDHCONF.CNF	Registered information of failure diagnosis handler
	AEHCONF.CNF	Registered information of abnormal exception handler

Configuration definition file is embedded and used as a part of source code of TRON Safe Kernel at the construction of TRON Safe Kernel.

Particular operation of configuration is implementation-defined. However following items should be observed.

(1) The value specified by configuration cannot be changed after the construction of TRON Safe Kernel.

(2) The value specified by configuration is surely checked by TRON Safe Kernel before using it. If the value of configuration is invalid, an abnormal exception occurs.

7.2 System Management Information

System management information is the numerical information to set various kinds of setting on TRON Safe Kernel.

List of the information specified by System management information is shown in 'Table 7-2 System management information List'.

Table 7-2 System management information List

Category	Name	Description	Range of value
System information	TS_LST_PRI	Lowest execution priority which can be specified (task priority)	16 to 250
	TS_HST_SPRI	Highest execution priority which can be specified (system priority)	-250 to -1
	TS_MAX_TSKTIM	Maximum continuous run time of task	1 to implementation-defined
	TS_MAX_TEHTIM	Maximum continuous run time of time event handler	1 to implementation-defined
	TS_MAX_DMNTIM	Maximum continuous run time of domain	1 to implementation-defined (*1)
	TS_MAX_DSPTIM	Maximum continuous run time of dispatch disabled	1 to implementation-defined
	TS_MAX_INTTIM	Maximum continuous run time of interrupt handler	1 to implementation-defined
	TS_MIN_SCHTIM	Minimum interval time of scheduler	1 to implementation-defined (*7)
	TS_MAX_SCHTIM	Maximum interval time of scheduler	1 to implementation-defined (*7)
	TS_MAX_SDMN	Number of Safety domain	1 to implementation-defined (*8)
	TS_MAX_NDMN	Number of Normal domain	0 to implementation-defined (*8)
	TS_CONF_DLYCHK	Specify delayed check of configuration information	1: Delayed check 0: Check at starting (*9)
	TS_SYSTIM_MET	System timer method	1: Timer variable interval time method 0: Timer fixed interval time method (*10)
	TS_STK_SIZE_INH	Dedicated stack size for interrupt	1 to UINT_32MAX(*3) (*11)
	TS_STK_SIZE_AEH	Dedicated stack size for abnormal exception	1 to UINT_32MAX(*3) (*11)

	TS_STK_SIZE_FDH	Dedicated stack size for fault diagnosis	1 to UINT_32MAX(*3) (*11)
	TS_TOTAL_MBFSZ	Total memory size of memory for buffer	1 to implementation-defined
	TS_MAX_MRGN	Maximum number of memory area in memory-map management information	1 to INT_16MAX(*2)
	TS_MAX_STACK	Maximum number of stack for each domain	1 to INT_16MAX(*2)
	TS_MAX_TMOUT	Maximum timeout time	1 to implementation-defined
Object information (common)	TS_MAX_SEMCNT	Number of maximum semaphore resource	1 to INT_16MAX(*2)
	TS_MAX_WUPCNT	Number of maximum request of task wakeup	1 to INT_16MAX(*2)
	TS_MAX_SUSCNT	Number of maximum request of waiting-suspended	1 to INT_16MAX(*2)
	TS_MAX_CYCTIM	Maximum time interval to start cyclic handler	1 to UINT_32MAX(*3)
	TS_MAX_CYCPHS	Maximum start-up phase time of cyclic handler	1 to UINT_32MAX(*3)
	TS_MAX_ALMTIM	Maximum start-up time of alarm handler	1 to UINT_32MAX(*3)
Object information (System domain)	TS_MAX_TSK_SY	Maximum number of task (System domain)	1 to INT_16MAX(*2)
	TS_MAX_SEM_SY	Maximum number of semaphore (System domain)	0 to INT_16MAX(*2)
	TS_MAX_FLG_SY	Maximum number of event flag (System domain)	0 to INT_16MAX(*2)
	TS_MAX_MTX_SY	Maximum number of mutex (System domain)	0 to INT_16MAX(*2)
	TS_MAX_MBF_SY	Maximum number of message buffer (System domain)	0 to INT_16MAX(*2)
	TS_MAX_CYC_SY	Maximum number of cyclic handler (System domain)	0 to INT_16MAX(*2)
	TS_MAX_ALM_SY	Maximum number of alarm handler (System domain)	0 to INT_16MAX(*2)
	TS_MAX_MSZ_SY	Maximum message size (System domain)	1 to INT_16MAX(*2)
Object	TS_MAX_TSK_SA	Maximum number of task (Safety	1 to INT_16MAX(*2)

information (Safety domain)		domain)	
	TS_MAX_SEM_SA	Maximum number of semaphore (Safety domain)	0 to INT_16MAX(*2)
	TS_MAX_FLG_SA	Maximum number of event flag (Safety domain)	0 to INT_16MAX(*2)
	TS_MAX_MTX_SA	Maximum number of mutex (Safety domain)	0 to INT_16MAX(*2)
	TS_MAX_MBF_SA	Maximum number of message buffer (Safety domain)	0 to INT_16MAX(*2)
	TS_MAX_CYC_SA	Maximum number of periodical handler (Safety domain)	0 to INT_16MAX(*2)
	TS_MAX_ALM_SA	Maximum number of alarm handler (Safety domain)	0 to INT_16MAX(*2)
	TS_LST_DPRI_SA	Lowest task priority which can be specified (Safety domain)	16 to TS_LST_PRI (*4)
	TS_HST_DPRI_SA	Highest task priority which can be specified (Safety domain)	16 to TS_LST_PRI (*4)
	TS_MAX_MSZ_SA	Maximum message size (Safety domain)	1 to INT_16MAX(*2)
Object information (Normal domain)	TS_MAX_TSK_NA	Maximum number of task (Normal domain)	1 to INT_16MAX(*2)
	TS_MAX_SEM_NA	Maximum number of semaphore (Normal domain)	0 to INT_16MAX(*2)
	TS_MAX_FLG_NA	Maximum number of event flag (Normal domain)	0 to INT_16MAX(*2)
	TS_MAX_MTX_NA	Maximum number of mutex (Normal domain)	0 to INT_16MAX(*2)
	TS_MAX_MBF_NA	Maximum number of message buffer (Normal domain)	0 to INT_16MAX(*2)
	TS_MAX_CYC_NA	Maximum number of cyclic handler (Normal domain)	0 to INT_16MAX(*2)
	TS_MAX_ALM_NA	Maximum number of alarm handler (Normal domain)	0 to INT_16MAX(*2)
	TS_LST_DPRI_NA	Lowest task priority which can be specified (Normal domain)	16 to TS_LST_PRI (*5)
	TS_HST_DPRI_NA	Highest task priority which can be specified (Normal domain)	16 to TS_LST_PRI (*5)
	TS_MAX_MSZ_NA	Maximum message size (Normal domain)	1 to INT_16MAX(*2)

Subsystem management	TS_MAX_SSY	Maximum registered number of subsystem	0 to 255
	TS_LST_SSYPRI	Lowest priority of subsystem which can be specified	16 to 255
	TS_MAX_SSYFNO	Maximum number of function code of subsystem	1 to 4095
	TS_MAX_SVCPARA	Maximum number of extended SVC parameter	4 to 255
Device management (Safety API)	TS_MAX_DEV_S	Maximum registered number of device (Safety API)	0 to INT_16MAX(*2)
	TS_MAX_OPDEV_S	Maximum number to open device (Safety API)	1 to INT_16MAX(*2)
	TS_MAX_RQDEV_S	Maximum number of device request(Safety API)	1 to INT_16MAX(*2)
	TS_DEVT_MBFSZ_S	Message buffer size for event notification (Safety API)	0 to INT_16MAX(*2,*6)
	TS_DEVT_MBFMAX_S	Maximum message length of message buffer for event notification (Safety API)	0 to INT_16MAX(*2)
Device management (Normal API)	TS_MAX_DEV_N	Maximum registered number of device(Normal API)	0 to INT_16MAX(*2)
	TS_MAX_OPDEV_N	Maximum number to open device(Normal API)	1 to INT_16MAX(*2)
	TS_MAX_RQDEV_N	Maximum number of device request(Normal API)	1 to INT_16MAX(*2)
	TS_DEVT_MBFSZ_N	Message buffer size for event notification (Normal API)	0 to INT_16MAX(*2,*6)
	TS_DEVT_MBFMAX_N	Maximum message length of message buffer for event notification (Normal API)	0 to INT_16MAX(*2)
Interrupt management	TS_MAX_INTNO	Maximum number of interrupt handler	1 to INT_16MAX(*2)
Failure diagnosis	TS_CHK_CB	Diagnosis of control block	1:valid,0:invalid
	TS_CHK_SYSTIM	Diagnosis of system timer interrupt handler	1:valid,0:invalid
	TS_CHK_MBF	Diagnosis of message buffer transmission error	1:valid,0:invalid
	TS_MAX_FDH	Maximum number of failure diagnosis handler	0 to INT_16MAX(*2)

	TS_MAX_FDHCYC	Maximum time interval of failure diagnosis handler	1 to UINT_32MAX(*3)
Abnormal exception	TS_MAX_AEX_FD	Maximum number of abnormal exception by failure diagnosis	0x00008000 to 0x0000FFFF
	TS_MAX_AEX_UD	Maximum number of user-defined abnormal exception	0x00010000 to 0x0001FFFF

*1 TS_MAX_DMNTIM should be larger than TS_MAX_TSKTIM and TS_MAX_TEHTIM.

*2 'INT16_MAX' indicates the maximum value (32767) of 16-bit width signed integer. However, 'INT16_MAX' is the maximum value permitted in the function specification it can be defined to a lower value in the implementation specification.

"UINT32_MAX" indicates the maximum value (4294967295) of 32-bit width unsigned integer. However "UINT32_MAX" is the maximum value permitted in the function specification it can be defined to a lower value in the implementation specification.

*4 The priority of 'TS_HST_DPRI_SA' should be the same as that of 'TS_LST_DPRI_SA' or higher (smaller value) than that of 'TS_LST_DPRI_SA'.

*5 The priority of 'TS_HST_DPRI_NA' should be the same as that of 'TS_LST_DPRI_NA' or higher (smaller value) than that of 'TS_LST_DPRI_NA'.

*6 In case of '0' message buffer for event notification is not created.

*7 The relationship between TS_MIN_SCHTIM and TS_MAX_SCHTIM should be 'TS_MIN_SCHTIM≦TS_MAX_SCHTIM'.

*8 The relationship between TS_MAX_SDMN and TS_MAX_NDMN should be 'TS_MAX_SDMN(number of Safety domain) + TS_MAX_NDMN(number of Normal domain) + 1 ≦ 127'.

*9 By specifying delayed check, configuration check at starting system is executed at starting domain.

*10 Method which can be specified is implementation-defined. Interval time of system timer interrupt is variable at timer variable interval time method. On the other hand, interval time of system timer interrupt is fixed at timer fixed interval time method.

*11 Dedecated stack is secured in the different area from '7.3.3Stack Management Information' for interrupt handler, abnormal exception handler and failure diagnosis handler.

Each value of the system management information is defined by the macro define (#define) of the C language.
The example of description is shown below.

```
/* system management information */
/* object management(system domain) */
#define  TSCF_MAX_TSK_SY    (100)    /* maximum number of task (system domain) */
#define  TSCF_MAX_SEM_SY    (50)     /* maximum number of semaphore (system domain) */
#define  TSCF_MAX_FLG_SY    (50)     /* maximum number of event flag (system domain) */
#define  TSCF_MAX_MTX_SY    (50)     /* maximum number of mutex (system domain) */
#define  TSCF_MAX_MBF_SY    (50)     /* maximum number of message buffer (system domain) */
#define  TSCF_MAX_CYC_SY    (50)     /* maximum number of cyclic handler(system domain) */
#define  TSCF_MAX_ALM_SY    (50)     /* maximum number of alarm handler (system domain) */
```

7.3 Memory Management Information

Memory management information is the information of configuration about the memory area.

Memory management information is shown below.

- Memory-map management information
- Stack management information
- Message buffer management information

7.3.1 Overview of memory area

The memory which is the operation target of TRON Safe Kernel is divided into plural memory areas and managed.

Each memory area has address indicating specific memory area and the information representing the attribute.

The contents of memory area information are show in 'Table 7-3 Memory area information List'.

Table 7-3 Memory area information List

Classification	Name	Description	Range of value
Memory area	mem_atr	Attribute of memory area	—
	mem_top	Initial address of memory area	Implementation-defined
	mem_bottom	End address of memory area	Implementation-defined

The address of memory area is specified by 'mem_top' and 'mem_bottom'.

'mem_top' is smaller than 'mem_bottom'.(mem_top < mem_bottom)

Memory area is categorized to code area, data area and stack area according to the attribute. Memory area attribute is shown in 'Table 7-4 Memory area attribute List'.

Attribute of 'TSMA_CDIS' is used in combination with other attributes. The area where 'TSMA_CDIS' can be used in combination with other attributes is implementation-defined.

Attribute of 'TSMA_NONE' shows that the memory area information is invalid (unused data).

Table 7-4 Memory area attribute List

Name	Description	Use
TSMA_CODE	Code area	The area where program code is stored
TSMA_DATA	Data area	The area where mainly statically allocated data is stored
TSMA_STACK	Stack area	The area which is used as stack while task or time event handler is running
TSMA_NONE	Unused data	Corresponding memory area information is invalid
TSMA_CDIS	Cache prohibition	The area where cache is not used for memory access

7.3.2 Memory-map Management Information

Memory-map is composed of several memory areas. Memory-map management information is the set of memory area information.

A memory-map management information is specified for each domain. Therefore number of memory-map management information is 'TS_MAX_SDMN + TS_MAX_NDMN + 1'.

The maximum number of memory area included in a memory-map management information is implementation-defined. However one or more code area and one or more data area and one stack area should be included. In other words, three or more memory areas are included in a memory-map management information.

Each value of memory-map management information is defined by C language macro definition (#define) and const type data (const type).

The description format of memory-map management information is shown as follows.

```
#define TS_MAX_MRGN          Maximum number of memory area in memory-map management information
TS_CONF_MMAP = {  /* Row of memory-map management information */
        /* First memory-map management information */
        {
                First memory area
                Second memory area
                /* Repetirion */
                TS_MAX_MRGN th memory area
        } ,
        /* Second memory-map management information */
        {
                /* Omitted */
        },

        /* Repetition(Omitted ) */

        /* (TS_MAX_SDMN + TS_MAX_NDMN + 1) th memry map management information */
        {
                /* Omitted */
        }
};
```

In memory-map management information described in 'TS_CONF_MMAP', memory-map number is numbered in described order. Memory-map number is from 1 to 'TS_MAX_SDMN + TS_MAX_NDMN + 1'.

Memory-map number which domain uses in domain management information described below is specified, and domain and the memory-map management information is associated.

Number of 'TS_MAX_MRGN' memory area informations are described in a memory-map management information. Unused memory area information is set with the TSMA_NONE attribute.

Description sample of memory-map management information is show as follows. In this example, there are three memory areas (one code area, one data area, one stack area) in a domain.

```
/* Memory-map management information */
#define TS_MAX_MRGN             3
                        /* Maximum number of memory area in memory-map management information */
TS_CONF_MMAP   = { /* Row of memory-map management information */
            /* First memory-map management information */
            {
                        {           /* Code area */
                                    TSMA_CODE,          /* Memory attribute */
                                    0x01000000, 0x01FFFFFF          /* Memory area address */
                        } .
                        {           /* Data area */
                                    TSMA_DATA,                      /* Memory attribute */
                                    0x02000000, 0x02FFFFFF          /* Memory area address */
                        } ,
                        {           /* Stack area */
                                    TSMA_STACK,                     /* Memory attribute */
                                    0x03000000, 0x03FFFFFF          /* Memory area address */
                        }
            } ;
            /* Second memory-map management information */
            {
                        /* Omitted */
            } ,

            /* Repetition(Omitted) */

            /* (TS_MAX_SDMN + TS_MAX_NDMN + 1)th memory-map management information */
            {
                        /* Omitted */
            }
};
```

7.3.3 Stack Management Information

Stack management information is configuration information about stack area.

As noted in the preceding paragraph, there is one stack area in each domain and it is used as stack when task and time event handler is running.
The use of stack area of each domain is shown in 'Table 7-5 Use of Stack area'.

Table 7-5 Use of Stack area

Domain type	Use of stack area
Safety domain	User stack of task and time event handler belonging to Safety domain
Normal domain	User stack of task and time event handler belonging to Normal domain
System domain	System stack of all the task and time event handler

In user stack of Normal domain, memory is allocated to the individual stack from stack area when creating task and time event handler. Therefore there is no stack management information in Normal domain.

Individual stack is allocated statically from stack area except for user stack of Normal domain. Individual stack is associated in creating task and time event handler. The static setting in stack management information is described.
A stack management information is specified for each domain except for Normal domain. Therefore there are number of 'TS_MAX_SDMN + 1' stack management information

Stack size used in corresponding domain is described at stack management information.
Maximum number of stack used in a domain is defined as 'TS_MAX_STACK'. Therefore, 1 stack management information consists of TS_MAX_STACK stack size.

In stack management information each value is defined at C language macro definition (#define) and const type data (const type)
Description format of stack management information is shown as follows.

```
#define TS_MAX_STACK            Maximum number of stack for each domain

TS_CONF_STACK = { /* Row of stack management information
            /* First stack management information */
            { Row of stack size },
            /* Second stack management information */
            { Row of stack size },
            /* Repetition(Omitted) */

            /* (TS_MAX_SDMN + 1)th stack management information */
            { Row of stack size }
```

}

In stack management information stack management number is assigned in the described order. Stack management number is the value of 1 to 'TS_MAX_SDMN + 1'.

In the domain management information described below, stack management number which domain uses is specified and domain and stack management information is associated.

Stack resource number is assigned in the described order to the stack size described in the stack management information. Stack resource number is the value of 1 to 'TS_MAX_STACK'.

If actually used resource number of stack is less than 'TS_MAX_STACK', unused value of stack size becomes '0'.

Unused stack whose stack size is '0' is put in the rear of the arrangement of the row of stack size.

In creating task and time event handler, used stack resource number is specified and task, time event handler and stack are associated.

Description sample of stack management information is shown as below.

```
/* Stack management information */
#define TS_MAX_STACK        10        /* Maximum value of stack for each domain */
TS_CONF_STACK = { /* Row of stack management information */
        /* First stack management information */
        { 1024, 1024, 1024, 1024, 512, 512, 256, 256, 0, 0 },
        /* Second stack management information */
        { 512, 512, 512, 512, 512, 512, 256, 256, 256, 256 },

        /* Repetition (Omitted)

        /* (TS_MAX_SDMN + 1)th stack management information */
        { 512, 512, 512, 512, 512, 512, 0, 0, 0, 0 },
}
```

7.3.4 Message Buffer Management Information

Message buffer uses memory for buffer for communication. Memory for buffer is allocated statically in the data area of system domain. Message buffer management information describes the information to allocate the memory for buffer of the message buffer.

Message buffer management information is composed of total memory size of memory for buffer and size of memory for buffer for each message buffer.

There are (TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA), as a total of message buffers for buffer memory size of each message buffer

In message buffer management information each value is defined at C language macro definition (#define) and const type data

(const type)

Description format of message buffer management information is shown as follows.

```
#define TS_TOTAL_MBFSZ        Total memory size of memory for buffer

TS_CONF_MBFSZ = {             /* Row of size of memory for buffer */
        First size of memory for buffer (for event notification message buffer of system default (for Safety API))
        Second size of memory for buffer (for event notification message buffer of system default (for Normal API))
        Third size of memory for buffer
        /* Repetition (Omitted)

        Size of (TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA)th memory for buffer
        }
```

Message buffer resource number is assigned in the described order to the size of memory for buffer. Message buffer resource number is the value of 1 to 'TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA'.

If actually used resource number of memory for buffer is less than 'TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA', unused value of size of memory for buffer becomes '0'.

Unused memory for buffer whose memory size is '0' is put in the rear of the arrangement of the row of size of memory for buffer.

In creating message buffer, message buffer resource number is specified and message buffer and message buffer management information are associated.

Memory size of event notification message buffer of system default of device management function is specified to first and second size of memory for buffer. 'TS_DEVT_MBFSZ_S' and 'TS_DEVT_MBFSZ_N' are specified to memory size. Refer to '4.1.7 Overview of TRON Safe Kernel' about the detail of event notification message buffer.

Description sample of message management information is shown as below. In the sample total number (TS_MAX_MBF_SY + TS_MAX_MBF_SA + TS_MAX_MBF_NA) of message buffer is '6'.

```
/* Message buffer management information */
#define TS_TOTAL_MBFSZ


TS_CONF_MBFSZ = {                /*  Row of size of memory for buffer */
        TS_DEVT_MBFSZ_S,   /* size of first memory for buffer */
        TS_DEVT_MBFSZ_N ,  /* size of second memory for buffer */
        2048,       /* size of third memory for buffer */
        512,        /* size of fourth memory for buffer */
        512,        /* size of fifth memory for buffer */
        0           /* size of sixth memory for buffer */
}
```

7.4 Domain management information

The domain management information is the numeric information to specify the various kinds of setting of each domain.

Domain management information is composed of following items.

- System domain management information
- Safety domain management information
- Normal domain management information

7.4.1 Overview of domain management information

Domain has individually domain management information shown in ‘Table 7-6 Domain management information List’.

Table 7-6 Domain management information List

Name	Description	Range of value
TDmnAtr	Domain attribute	(*7)
TMemMap	Memory map number	1 to Maximum value of memory map number(*1)
TStackRes	Stack resource number	1 to Maximum value of stack resource number(*2)
TMaxTskId	Maximum number of task	1 to Maximum number which can be set(*3)
TMaxSemId	Maximum number of semaphore	0 to Maximum number which can be set(*3)
TMaxFlgId	Maximum number of event flag	0 to Maximum number which can be set(*3)
TMaxMtxId	Maximum number of mutex	0 to Maximum number which can be set(*3)
TMaxMbfId	Maximum number of message buffer	0 to Maximum number which can be set(*3)
TMaxCycId	Maximum number of periodical handler	0 to Maximum number which can be set(*3)
TMaxAlmId	Maximum number of alarm handler	0 to Maximum number which can be set(*3)
TLstTskPri	Lowest task priority which can be specified	Range which can be set at domain(*4)
THstTskPri	Highest task priority which can be specified	Range which can be set at domain(*4)
TMaxTskTim	Maximum continuous run time of task	1 to TS_MAX_TSKTIM
TMaxTehTim	Maximum continuous run time of time event handler	1 to TS_MAX_TSKTIM
TMaxDmnTim	Maximum continuous run time of domain	1 to TS_MAX_DMNTIM(*5)
TMaxDspTim	Maximum continuous run time of dispatch disabled	1 to TS_MAX_DSPTIM
TIniTsk	Information to create initial task	*6

*1 Memory map used by domain is specified. Refer to ‘7.3.2 Memory-map Management Information ’ about the detail of memory map.

*2 Stack management information used by domain is specified. However it is ignored in Normal domain because there is no stack management information in Normal domain. Refer to ‘7.3.3 Stack Management Information ’ about the detail of stack management information.

*3 About maximum number which can be set, the total value in the same type of domain should be less than the maximum number of the object information of system management information. For example, the total value of ‘TMaxTskId’ of all Safety domains is less than the value of ‘TS_MAX_TSK_SA’ of system management information.

*4 ‘THstTskPri’ and ‘TLstTskPri’ of application domain (Safety domain and Normal domain) are within the range of specified priority at the system management information. Those of System domain are possible to specify the range from 1 to ‘TS_Lst_PRI’. ‘THstTskPri’(maximum priority) is the same as ‘TLstTskPri’(minimum priority) or is higher (smaller value) than ‘TLstTskPri’(minimum priority).

*5 ‘TMaxDmnTim’ should be larger than ‘TMaxTskTim’ and ‘TMaxTehTim’.

*6 It conforms to API parameter (T_CTSK). However it is ignored because there is no initial task which should be defined at configuration in System domain.
Refer to ‘8.2.1.1 ts_cre_tsk/tn_cre_tsk – Create Task’. about the detail of ‘T_CTSK’

*7 Domain attribute is specified. The attribute which can be specified is only ‘attribute to execute at starting’ (TA_START). And ‘attribute to execute at starting’ is valid only in Safety domain and it is ignored in System domain and Normal domain.

7.4.2 System domain management information

There is only one System domain. Therefore content of system domain management information is same as domain information.

System domain management information is defined as C language const data (const type).

The description format of System domain management information is shown below.

```
SYS_DMN_CONF   = { /* System domain management information */
        .TDmnAtr        = Domain attribute
        .TMemMap        = Memory map number
        .TStackRes      = Stack resource number
        .TMaxTskId      = Maximum number of task
        .TMaxSemId      = Maximum number of semaphore
        .TMaxFlgId      = Maximum number of event flag
        .TMaxMtxId      = Maximum number of mutex
        .TMaxMbfId      = Maximum number of message buffer
        .TMaxCycId      = Maximum number of periodical handler
        .TMaxAlmId      = Maximum number of alarm handler
        .TLstTskPri     = Lowest task priority which can be specified
        .THstTskPri     = Highest task priority which can be specified
        .TMaxTskTim     = Maximum continuous run time of task
        .TMaxTehTim     = Maximum continuous run time of time event handler
        .TMaxDmnTim     = Maximum continuous run time of domain
        .TMaxDspTim     = Maximum continuous run time of dispatch disabled
        .TIniTsk        = Information to create initial task(invalid at System domain)
};
```

The information to create initial task is ignored because there is no initial task which should be defined in System domain.

As System domain is definitely executed at starting system, 'attribute to execute at starting' of domain attribute is ingored.

The example of description of System domain management information is shown below.

```
/* System domain management information */
SYS_DMN_CONF = {
        .TDmnAtr        = (ATR)0,   /* Domain attribute */
        .TMemMap        = 1,        /* Memory map number */
        .TStackRes      = 1,        /* Stack resource number */
        .TMaxTskId      = 20,       /* Maximum number of task */
        .TMaxSemId      = 10,       /* Maximum number of semaphore */
        .TMaxFlgId      = 5,        /* Maximum number of event flag */
        .TMaxMtxId      = 5,        /* Maximum number of mutex */
        .TMaxMbfId      = 5,        /* Maximum number of message buffer */
        .TMaxCycId      = 5,        /* Maximum number of periodical handler */
        .TMaxAlmId      = 5,        /* Maximum number of alarm handler */
        .TLstTskPri     = 250,      /* Lowest task priority which can be specified */
        .THstTskPri     = 1,        /* Highest task priority which can be specified */
        .TMaxTskTim     = 500,      /* Maximum consective execution time of task */
        .TMaxTehTim     = 500,      /* Maximum consective execution time of time event handler */
        .TMaxDmnTim     = 1000,     /* Maximum consective execution time of domain */
        .TMaxDspTim     = 1000,     /* Maximum consective execution time of dispatch
                                    prohibitation */
        .TIniTsk        =           /* Information to create initial task(invalid at System domain) */
                        { (ATR)0, (FP)0, (PRI)0, (RELTIM)0, {0,0,0,0,0,0,0,0}}
};
```

7.4.3 Safety domain management information

There exists the number of 'TS_MAX_SDMN' defined by System management information in Safety domain, so the number of Safety domain management information is 'TS_MAX_SDMN' rows of domain information
Safety domain management information is defined as C language const type data (const type).
The description format of Safety domain management information is shown below.

```
SAFE_DMN_CONF = { /* Safety domain management information */
        /* Domain information of first Safety domain */
        {
                .TDmnAtr        = Domain attribute
                .TMemMap        = Memory map number
                .TStackRes      = Stack resource number
                .TMaxTskId      = Maximum number of task
                .TMaxSemId      = Maximum number of semaphore
                .TMaxFlgId      = Maximum number of event flag
```

```
        .TMaxMtxId        = Maximum number of mutex
        .TMaxMbfId        = Maximum number of message buffer
        .TMaxCycId        = Maximum number of periodical handler
        .TMaxAlmId        = Maximum number of alarm handler
        .TLstTskPri       = Lowest task priority which can be specified
        .THstTskPri       = Highest task priority which can be specified
        .TMaxTskTim       = Maximum continuous run time of task
        .TMaxTehTim       = Maximum continuous run time of time event handler
        .TMaxDmnTim       = Maximum continuous run time of domain
        .TMaxDspTim       = Maximum continuous run time of dispatch disabled
        .TIniTsk          = Information to create initial task
    },
    /* Domain information of second Safety domain */
    {
        /* omission */
    },

    /* repetition (omission) */
    /* 'TS_MAX_DEV_A + TS_MAX_DEV_N'th domain information of Safety domain  */
    {
        /* omission */
    }
};
```

Domain ID corresponding to domain information in the described order in the Safety domain management information is allocated. Safety domain ID begins at '2' so maximum number of Safety domain ID is 'TS_MAX_SDMN＋1'.

```
/* Safety domain management information */
SAFE_DMN_CONF = {
	/* Domain information of first Safety domain */
	{
		.TDmnAtr		= TA_START,	/* Domain attribute */
		.TMemMap		= 2,		/* Memory map number */
		.TStackRes		= 2,		/* Stack resource number */
		.TMaxTskId		= 20,		/* Maximum number of task */
		.TMaxSemId		= 10,		/* Maximum number of semaphore */
		.TMaxFlgId		= 5,		/* Maximum number of event flag */
		.TMaxMtxId		= 5,		/* Maximum number of mutex */
		.TMaxMbfId		= 5,		/* Maximum number of message buffer */
		.TMaxCycId		= 5,		/* Maximum number of periodical handler */
		.TMaxAlmId		= 5,		/* Maximum number of alarm handler */
		.TLstTskPri		= 100,		/* Lowest task priority which can be specified */
		.THstTskPri		= 16,		/* Highest task priority which can be specified */
		.TMaxTskTim		= 500,		/* Maximum consective execution time of task */
		.TMaxTehTim		= 500,
					/* Maximum consective execution time of time event handler */
		.TMaxDmnTim		= 1000,		/* Maximum consective execution time of domain */
		.TMaxDspTim		= 1000,
				/* Maximum consective execution time of dispatch prohibitation */
		.TIniTsk		=		/* Information to create initial task */
		{
			(ATR)0,			/* Task attribute */
			(FP)SDTSK_MAIN,		/* Task start-up address */
			(PRI)16,		/* Task start-up priority */
			(RELTIM)100,		/* Maimum execution time */
			/* Object name */
			{'I','N','I','T','A','S','K',0}
		}
	},
	/* Domain information of second Safety domain */
	{
		/* Omission */
	},

	/* Repetition (Omission) */

	/* Domain information of 'TS_MAX_SDMN'th Safety domain */
	{
		/* Omission */
	}
};
```

7.4.4 Normal domain management information

There exists the number of ‘TS_MAX_ NDMN’ Normal domains defined by system management information. , so Normal domain management information is row of number of ‘TS_MAX_NDMN’

Normal domain management information is defined as C language const type data (const type).
The description format of Normal domain management information is shown below.

```
NORM_DMN_CONF = {/* Normal domain management information */
        /* Domain information of first Normal domain */
        {
                /* Omission (Same as Safety domain) */
        },
        /* Domain information of second Normal domain */
        {
                /* Omission : */
        },

        /* repetition (omission) */

        /* Domain information of ‘TS_MAX_INTNO’th Normal domain */
        {
                /* Omission */
        }
};
```

Domain ID corresponding to domain information in the described order in the normal management information is allocated.
Normal domain ID begins at ‘TS_MAX_SDMN＋2’ because it begins at the next value of maximum ID of Safety domain.
Maximum Normal domain ID is ‘TS_MAX_SDMN＋TS_MAX_NDMN + 1’.
As Normal domain is started by task of System domain or Safety domain, ‘attribute to execute at starting’ of domain attribute is ingored.

Description sample of Normal domain management information is shown as below.

```
/* Normal domain management information /
NORM_DMN_CONF = {
        /* Domain information of first normal domain */
        {
                .TDmnAtr        = (ATR)0, /* Domain attribute */
                .TMemMap        = 3,      /* Memory map number */
                .TStackRes      = 0,      /* Stack resource number (invalid in Normal domain)
                .TMaxTskId      = 20,     /* Maximum number of task */
                .TMaxSemId      = 10,     /* Maximum number of semaphore */
                .TMaxFlgId      = 5,      /* Maximum number of event flag */
                .TMaxMtxId      = 5,      /* Maximum number of mutex */
                .TMaxMbfId      = 5,      /* Maximum number of message buffer */
                .TMaxCycId      = 5,      /* Maximum number of periodical handler */
                .TMaxAlmId      = 5,      /* Maximum number of alarm handler */
                .TLstTskPri     = 256,    /* Lowest task priority which can be specified */
                .THstTskPri     = 50,     /* Highest task priority which can be specified */
                .TMaxTskTim     = 500,    /* Maximum consecive execution time of task */
                .TMaxTehTim     = 500,
                                /* Maximum consecive execution time of time event handler */
                .TMaxDmnTim     = 1000,   /* Maximum consecive execution time of domain */
                .TMaxDspTim     = 1000,

                                /* Maximum consecive execution time of dispatch prihibitaton */
                .TIniTsk        =         /* Information to create initial task */
                {
                        (ATR)0,           /* Task attribute */
                        (FP)NMTSK_MAIN,   /* Task start-up address */
                        (PRI)50,          /* Task start-up priority */
                        (RELTIM)100,      /* Maximum execution time */
                        /* Object name */
                        {'I','N','I','T','A','S','K',0}
                }
        },
        /* Domain information of second Normal domain */
        {
                /* Omission */
        },

        /* Repetition (Omission) */

        /* Domain information of 'TS_MAX_NDMN'th Safety domain */
        {
                /* Omission */
        }
};
```

7.5 Registeration information of User-defined system software

The registration information of user-defined system software is the information of user-defined system software registered at the initial state of TRON Safe Kernel. It is described by the Initial definition API.

7.5.1 Initial definition API of registration information of user-defined system software

Initial definition API described at user-defined system software registration information is shown in 'Table 7-7 Initial definition API List of registration information of user-defined system software'.

Table 7-7 Initial definition API List of registration information of user-defined system software

Classification	API Name	Function
Initial definition API	TS_DEF_DEV	Registration of a device driver
	TS_DEF_INT	Registration of an interrupt handler
	TS_DEF_SSY	Registration of a subsystem
	TS_DEF_FDH	Registration of a failure diagnosis handler
	TS_DEF_AEH	Registration of an abnormal exception hander

The explanation of each Initial definition API is shown as below.

7.5.1.1 TS_DEF_DEV – Registration of device driver

C language interface

[Initial definition API] void TS_DEF_DEV(CONST T_DDEV *pk_ddev);

Parameter

CONST T_DDEV* pk_ddev Device driver registration information

```
typedef struct st_ddev {
Device driver registration information
        UB      devnm[8];       /* Physical device name */
        ATR     drvatr;         /* Device driver attribute */
        INT     nsub;           /* Sub-unit number */
        INT     blksz;          /* Unique data block size (-1:unknown) */
        FP      initfn,         /* Initial function address */
        FP      openfn;         /* Open function address */
        FP      closefn;        /* Close function address */
        FP      execfn;         /* Starting processing function address */
        FP      waitfn;         /* Waiting for completion function address */
        FP      cancelfn;       /* Request invalid function address */
        FP      abortfn;        /* Aborting processing function address */
        FP      eventfn;        /* Event function address */
} T_DDEV;
```

Description

Device driver is registered according to the specified value by 'pk_ddev'. If NULL(0) is specified to 'pk_ddev', error occurs. Physical device name of registered device driver is specified by 'dennm'. Device driver attribute is in accordance with '4.1.2 Device Driver Attributes'.

Sub-unit number of device driver is specified by 'nsub'. The value of '0' to '254' can be specified as the number of sub-unit. However actually valid number of sub-unit is defined for each device driver.

Unique data block size by number of bytes is set by 'blksz'. In case of disc device, 'blksz' is block size. The size of serial line is 1 byte. In case of device which has no qnique data, 'blksz' is zero. In case that block size is inknown like unformatted disc, 'blksz' is '-1'. In case of 'blksz≦0', user can not access to qnique data. In case that user accesses to unique data by 'ts/tn_rea_dev' or 'ts/tn_wri_dev' , 'size * blksz' should be the area size to access, that is the size of 'buf'.

Execution start address of driver interface function of registered device driver is specified by 'initfn'、' openfn'、' closefn'、' execfn'、' waitfn'、'cancelfn'、' abortfn' and 'eventfn'. Detail of device driver function is shown in '4.1.5 Device Driver '

If device driver is registered, device ID is allocated in the order of invoking Initial definition API. Device driver ID begins at '1'. Maximum device ID is total of registerable device driver and 'TS_MAX_DEV_A + TS_MAX_DEV_N'.

The registered device driver can be operated by the API of device management function after starting TRON Safe Kernel. Basic operation is specified by using device name specified by this Initial definition API. And each kinds of information of device including device ID can be get by API to get device information (ts_ref_dev/tn_ref_dev).

This API does not return value. If error occurs while executing, an abnormal exception occurs.

Description sample

```
TS_DEF_DEV ( &(T_DEV) {
        { 'i', 'o', 'p', 'a', 0, 0, 0, 0 },   /* Device name */
        TDA_SEV | TDA_OPENREQ ,              /* Device driver attribute */
        0 ,                                  /* Sub-unit number */
        1,                                   /* Unique data block size */
        fp_initfn,                           /* Initial function address */
        fp_openfn;                           /* Open function address */
        fp_closefn;                          /* Close function address */
        fp_execfn;                           /* Starting processing function address */
        fp_waitfn;                           /* Waiting for completion function address */
        fp_cancelfn;                         /* Request invalid function address */
        fp_abortfn;                          /* Aborting processing function address */
        fp_eventfn;                          /* Event function address */
} ) ;
```

And device driver interface function shall be defined separately.

7.5.1.2 TS_DEF_INT – Registration of interrupt handler

C language interface

[Initial definition API] void TS_DEF_INT(UINT dintno, CONST T_DINT* pk_dint);

Parameter

UINT	dintno	Interrupt handler number
CONST T_DINT*	pk_dint	Interrupt handler definition information

Interrupt handler definition information

```
typedef struct st_dint {
        ATR     intatr;         /* Interrupt handler attribute */
        FP      inthdr;         /* Interrupt handler start-up address */
        INT     priority        /* Execution priority */
        RELTIM  maxrtim;        /* Maximum continuous run time */
} T_DINT;
```

Description

This API registers an interrupt handler whose interrupt handler number is specified by 'dintno'.

The operation of this API is same as API of registration of interrupt handler (ts_def_int)

- Initial definition API does not return value. If error occurs while executing, an abnormal exception occurs. Possible error is equivalent to 'ts_def_int'.
- Initial definition API cannot register again an interrupt handler which has been registered. If an interrupt handler number which has been registered is specified, error occurs.
- Initial definition API cannot release an interrupt handler which has been registered. If NULL(0) is specified to 'pk_dint', error occurs.

Refer to '8.3.3.1 ts_def_int – Define Interrupt Handler' about the detail of API of registration of interrupt handler (ts_def_int.)

Description sample

```
TS_DEF_INT ( DINT_1, &(T_DINT){ TIA_MLTINT, inthdr, 10 , 100 } );
```

And following shall be defined.

```
#define  DINT_1          1          /* Interrupt handler number */
void     int1_hdr( UINT );          /* Interrupt handler */
```

7.5.1.3 TS_DEF_SSY – Registration of subsystem

C language interface

[Initial definition API] void TS_DEF_SSY(ID ssid, CONST T_DSSY* pk_dssy);

Parameter

ID	ssid	Subsystem ID
CONST T_DSSY*	pk_dssy	Pointer to subsystem definition information

Subsystem definition information

```
typedef struct st_dssy {
        ATR     ssyatr;         /* Subsystem attribute */
        PRI     ssypri;         /* Subsystem priority */
        INT     svcnum;         /* Function number */
        FP      svchdr_s;       /* Start-up address of extended SVC for Safety API */
        FP      svchdr_n;       /* Start-up address of extended SVC for Normal API */
        FP      initfn;         /* Address of initial function */
        FP      startupfn;      /* Address of start-up function */
        FP      cleanupfn;      /* Address of clean-up function */
        FP      eventfn;        /* Address of event processing function */
        FP      breakfn;        /* Address of break function */
} T_DSSY;
```

Description

This API registers a subsystem by using subsystem ID specified by 'ssid'.

The value of '1' to 'TS_MAX_SSY' can be specified as subsystem ID. This ID should not be duplicated to the ID of the other system.

The information of registered subsystem is specified by 'pk_dssy'.

Subsystem attribute is specified by 'ssyatr'. Detail is shown in '4.3.4 Subsystem Attributes'.

Subsystem priority is specified by 'ssypri'. The value of '1' to 'TS_LST_SSYPRI' can be specified.

The number of function code provided by subsystem is specified by 'svcnum'.

Start-up address of extended SVC for Safety API is specified by 'svchdr_s' and start-up address of extended SVC for Normal API is specified by 'svchdr_n'.

Detail of an extended SVC is described in '4.3.5 Extended SVC '. If the same address is specified to 'svchdr_s' and 'svchdr_n', an abnormal exception occurs as an error.

Start-up address of subsystem interface function is specified by 'initfn', 'startupfn', 'cleanupfn', 'eventfn' and 'breakfn'.

Detail of subsystem interface function is described in '4.3.6 Subsystem Interface'.

An extended SVC can be issued to the registered subsystem by using API of subsystem management function after start-up of TRON Safe Kernel.

This API does not return value. If error occurs while executing, an abnormal exception occurs.

Description sample

```
TS_DEF_SSY ( SSID_1,
        &(T_DSSY){
                TSA_SSSY ,          /* Subsystem attribute */
                32 ,                /* Subsystem priority */
                10 ,                /* Function number */
                fp_svchdr ,         /* Start-up address of extended SVC for Safety API */
                FP(0) ,             /* Start-up address of extended SVC for Normal API */
                fp_initfn ,         /* Initial function_address */
                fp_tartupfn ,       /* Start-up function_address */
                fp_cleanupfn,       /* Clean-up function_address */
                fp_eventfn;         /* Event processing function_address */
                fp_breakfn;         /* Break function_address */
        }
);
```

And following shall be defined.

```
#define SSID_1          1       /* Subsystem ID */
```

And each extended SVC handler and subsystem interface function shall be defined.

7.5.1.4 TS_DEF_FDH – Registration of failure diagnosis handler

C language interface

[Initial definition API] void TS_DEF_FDH(ID fdhid, CONST T_DFDH* pk_dfdh);

Parameter

ID	fdhid	Failure diagnosis hander ID
CONST T_DFDH*	pk_dfdh	Pointer to failure diagnosis handler definition information

Failure diagnosis handler definition information

```
typedef struct st_dfdh {
        ATR      fdhatr;          /* Failure diagnosis handler attribute */
        PRI      fdhpri;          /* Execution priority of failure diagnosis handler */
        RELTIM   fdhcyc;          /* Start-up interval of failure diagnosis handler */
        FP       fdhdr;           /* Start-up address of failure diagnosis handler */
        RELTIM   maxrtim;         /* Maximum continuous run time */
} T_DFDH;
```

Description

This API registers a failure diagnosis handler by failure diagnosis handler ID specified by ‘fdhid’.

Information of registered failure diagnosis hander is specified by ‘pk_dfdh’.Failure diagnosis handler attribute is specified by ‘fdhatr’. Detail is shown in ‘4.4.3 Attributes of Failure Diagnosis ’.

Execution priority of failure diagnosis handler is specified by ‘fdhpri’. The value of range of task priority can be specified.
The start-up interval is specified by ‘fdhcyc’. The value of ‘1’ to ‘TS_MAX_FDHCYC’ can be specified.
Start-up address of failure diagnosis handler is specified by ‘fdhdr’. Detail is shown in ‘4.4.6 Execution Program for Failure Diagnosis ’.
The upper limit of consecutive execution time of failure diagnosis handler is specified by ‘maxrtim’. The value of ‘1’ to ‘TS_MAX_DMNTIM’ can be specified. Detail is shown in ‘4.4.7 Time Protection of Failure Diagnosis ’.

Registered failure diagnosis handler is executed at specified timing after starting TRON Safe Kernel. Detail is shown in ‘4.4.5 Start of Failure Diagnosis ’.

This API does not return value. If error occurs while executing, an abnormal exception occurs.

Description sample

```
TS_DEF_FDH( FDHID_1,
        &(T_DFDH){
                TFA_CYC ,         /* Failure diagnosis handler attribute */
                16 ,              /* Execution priority of failure diagnosis handler */
```

```
            100 ,               /* Start-up interval of failure diagnosis handler */
            fp_fdhdr,           /* Start-up address of failure diagnosis handler */
            1000                /* Maximum continuous run time */
      }
);
```

And following shall be defined.

```
#define     FDHID_1           1           /* Failure diagnosis handler ID */
```

And execution function of failure diagnosis handler shall be defined.

7.5.1.5 TS_DEF_AEH – Registration of abnormal exception handler

C language interface

[Initial definition API] void TS_DEF_AEH(UINT aexpno, CONST T_DAEH* pk_daeh);

Parameter

UINT	aexpno	abnormal exception number
CONST T_DAEH*	pk_daeh	Pointer to abnormal exception handler definition information

Abnormal exception handler definition information

```
typedef struct st_dfdh {
        ATR     aehatr;         /* Abnormal exception handler attribute */
        FUNCP   aehdr;          /* Start-up address of abnormal exception handler */
} T_DAEH;
```

Description

This API resisters an abnormal exception handler by an abnormal exception number specified by 'aexpno'.
This Initial definition API cannot release or register again the abnormal exception handler which has been registered. If an abnormal exception number which has been registered is specified, error occurs.

An abnormal exception handler attribute is specified by 'aehatr'. Detail is shown in '6.2.4 Attribute of Abnormal exception handler'.
Start-up address of an abnormal exception handler is specified by 'aehdr'. Detail is shown in '6.2.6 Execution program for Abnormal exception handler'.

This API does not return value. If error occurs while executing, an abnormal exception occurs.

Description sample

```
TS_DEF_AEH( AEXP_DIVZ,                  /* Abnormal exception number */
        &(T_DAEH){
                TAA_RETURN ,            /* Abnormal exception handler attribute */
                fp_aehdr,               /* Start-up address of abnormal exception handler */
        }
);
```

And execution function of an abnormal exception handler shall be defined.

8. API specification

8.1 Difference between Safety API and Normal API

Refer to ‘2.4.2 Safety API’ and ‘2.4.3 Normal API’.

8.2 API for TRON Safe Kernel/OS

8.2.1 API for Task Management Functions

Refer to ’3.1 Task Management’ about the detail of Task Management Functions.

8.2.1.1 ts_cre_tsk/tn_cre_tsk – Create Task

C Language Interface

[Safety API]	ID tskid = ts_cre_tsk(CONST T_CTSK *pk_ctsk);
[Normal API]	ID tskid = tn_cre_tsk(CONST T_CTSK *pk_ctsk);

Parameter

CONST T_CTSK*	pk_ctsk	Pointer to information about task creation

Information for creating a task

```
typedef struct st_ctsk {
        ATR     tskatr;         /* Task attribute  */
        FP      task;           /* Task start-up address  */
        PRI     itskpri;        /* Task start-up priority */
        UINT    ustkno;         /* Resource number of user stack */
        INT     ustksz;         /* User stack size (in bytes)
        UINT    sstkno;         /* Resource number of system stack */
        INT     sstksz;         /* System stack size (in bytes)
        RELTIM  maxrtim;        /* Maximum continuous run time  */
        UB      oname[8];       /* Object name */
} T_CTSK;
```

Return parameter

ID	tskid	Task ID
	or	Error Code

Error code

E_NOMEM	Insufficient memory error (Stack of specified size cannot be allocated from stack area)
E_LIMIT	Task count exceeds the system limit
E_RSATR	Reserved attribute error ('tskatr' is invalid or cannot be used), or the specified coprocessor does not exist
E_PAR	Parameter error (Address specified by 'pk_ctsk', 'task' and 'maxrtim', are invalid)
E_NOCOP	The specified coprocessor cannot be used (not installed, or abnormal operation detected)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_ONAME	Object name error (Specified object name has already been used in the domain)
E_MACV	Memory access privilege error (There is no access authority to 'pk_ctsk' or 'task', there is no target memory area, or the memory area specified by 'ustkno' or 'sstkno' is not a data area of the self domain.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API creates a task and assigns a task ID number.

After the task is created, it is initially in DORMANT state.

'**tskatr**' : Set a task attributes for indicating an operating condition of the task.

The system attribute part of tskatr is as follows

tskatr := [TA_ONAME]| [TA_COP0] | [TA_COP1] | [TA_COP2] | [TA_COP3] | [TA_FPU]

TA_ONAME	Set an object name
TA_COPn	Specifies use of the nth coprocessor (including floating point coprocessor or DSP)
TA_FPU	Specifies use of a floating point coprocessor (when a coprocessor specified in TA_COPn is a general-purpose FPU particularly for floating point processing and not dependent on the CPU)

```
#define TA_ONAME    (0x00000040U)    /* Set an object name */
#define TA_COP0     (0x00001000U)    /* Use ID =0 coprocessor */
#define TA_COP1     (0x00002000U)    /* Use ID =1 coprocessor  */
#define TA_COP2     (0x00004000U)    /* Use ID =2 coprocessor  */
#define TA_COP3     (0x00008000U)    /* Use ID =3 coprocessor  */
```

Task is described as a function in C language in the format below.

```
void    task_main( INT stacd );
```

'task' : Set a pointer to the function task start address described above.

Task start code :

The startup parameters passed to the task include the task startup code stacd specified in ‘tk_sta_tsk/tn_sta_tsk’.

A task can not or must not be terminated by a simple return from the function. A task definitely exits by issuing 'ts_ext_tsk/tn_ext_tsk' or 'ts_exd_tsk/tn_exd_tsk'.

If the function of a task is terminated by a simple return, , API exits its task (same operation as by issuing 'ts_ext_tsk/tn_ext_tsk') and then notifies the abnormality exeception.

'itskpri' : Set an initial priority for task starting. It is possible for its task priority to set a value between '1' and '250', lower value corresponds to higher priority. The priority which can be specified actually is defined as follows by the configuration for each

domain.

- The priority which can be specified for the task of System domain is ‘1’ to ‘TS_LST_PRI’.
- The priority which can be specified for the task of Safety domain is specified in the range of ‘TS_HST_DPRI_SA’ to ‘TS_LST_DPRI_SA’ for each domain.
- The priority which can be specified for the task of Normal domain is specified in the range of ‘TS_HST_DPRI_NA’ to ‘TS_LST_DPRI_NA’ for each domain.

Error code 'E_PAR' is returned, if setting higher priority than the maximum on the self domain.

Task has user stack and system stack. However the task of System domain has only system stack and does not have user stack. Therefore user stack specification is ignored for task creating of System domain.
The memory of user stack is allocated from the stack area of the domain which the task belongs to. The stack resource number specified by configuration is specified to ‘ustkno’. The size of user stack is specified to ‘ustksz’. However since memory area is allocated dynamically, the value of ‘utskno’ is ignored in Nomal API. In Safety API ‘ustksz’ should be less than the memory size of the stack resource specified by ‘ustkno’.
The memory of system stack is allocated from the stack area of System domain. Stack resource number specified by configuration is specified to ‘sstkno’. System stack size is specified to ‘sstksz’.’sstksz’ is less than the memory size of the stack resource specified by ‘sstkno’.
If the stack cannot be allocated for following reason, error ‘E_NOMEM’ occurs.

- Specified stack size is larger than the size of stack resource.
- Specified stack resource has been already used by task or time event handler.
- User stack cannot be allocated in Normal API (Insufficient size remaining of the user stack area)

'maxrtim' : Set the maximum consecutive execution time by milliseconds. Maximum value of 'maxrtim' corresponds to the maximum consecutive execution time on the self domain.
Error 'E_PAR' is notified, if setting longer time than the maximum consecutive execution time on the self domain.
Abnormal exception is notified, if task executing after passing the time set by 'maxrtim'. And then the task is suspended.
If ‘TA_ONAME’ is specified, ‘oname’ becomes effective.

'oname' : Set an object name of the task.
Set 'TA_ONAME' because of 'oname' being effective in case of setting it.

[Supplement]
TA_COPn definition has no portability because of hardware dependence, such as CPU and so on.
TA_FPU is used for floating point operation with portability in TA_COPn definition. For example, TA_FPU=TA_COP0 if floating point co-processor being TA_COP0. TA_FPU=0 if floating point coprocessor not used.

8.2.1.2 ts_del_tsk/tn_del_tsk – Delete Task

C Language Interface

[Safety API] ER ercd = ts_del_tsk(ID tskid);

[Normal API] ER ercd = tn_del_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the task specified in 'tskid' does not exist)
E_OBJ	Invalid object state (the task is not in DORMANT state)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

DescriptionDeletes the task specified in '**tskid**'This system call changes the state of the task specified in tskid from DORMANT state to NONEXISTENT state.

When this API is issued for a task not in DORMANT state, error code E_OBJ is returned.
This API cannot specify the invoking task. If the invoking task is specified, error code E_OBJ is returned since the invoking task is not in DORMANT state. The invoking task is deleted not by this API but by the 'ts_exd_tsk/tn_exd_tsk' system call.

8.2.1.3 ts_sta_tsk/tn_sta_tsk – Start Task

C Language Interface

[Safety API]	ER ercd = ts_sta_tsk(ID tskid, INT stacd);
[Normal API]	ER ercd = tn_sta_tsk(ID tskid, INT stacd);

Parameter

ID	tskid	Task ID
INT	stacd	Task start code

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number (‘tskid’ is invalid or cannot be used)
E_NOEXS	Object does not exist (the task specified in ‘tskid’ does not exist)
E_OBJ	Invalid object state (the task is not in DORMANT state)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

Starts the task specified in 'tskid'.This changes the state of the target task from DORMANT state to READY state. The main function for the target task is executed after changing to RUNNING state.

Parameters to be passed to the task when it starts are set in ‘stacd. Its parameter is passed to the main function for the task.

The task priority when it starts is the task start-up priority (‘itskpri’) specified when the started task was created. If the task priority is changed after creation between 'ts_chg_pri/tn_chg_pri' during DORMANT state, the task priority corresponds to its setting value.

Start requests by this API are not queued. If this API is issued while the target task is in a state other than DORMANT state, error code E_OBJ is returned to the calling task.

The following settings for the task in DORMANT state are not cleared when starting the task.

- Task priority (Task start-up priority)

- Disabled wait factors

8.2.1.4 ts_ext_tsk/tn_ext_tsk – Exit Task

C Language interface

[Safety API]	void ts_ext_tsk(void);
[Normal API]	void tn_ext_tsk(void);

Parameter

None

Return parameter

None (Not returned to the context issuing API)

Error code

None (Notify by exception if error detected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Explanation

This API terminates the invoking task normally and transited it to dormant state (DORMANT).

Information in TCB such as task priority, etc. are set to the initial values at the task returned to on dormant state (DORMANT).
For example, if a task with priority changed by 'ts_chg_pri'/'tn_chg_pri' is terminated, its priority is resumed to the task start-up priority ('itskpri') set by 'ts_cre_tsk'/'tn_cre_tsk'. Therefore, if the task started again by 'ts_sta_tsk'/'tn_sta_tsk', its priority is different from the one at executing this API in some cases.

This API is NOT returned to the context at issuing even if detecting any errors. Therefore, error code cannot be returned directly as a return parameter of API. This API notifies an abnormality exception if detecting any errors.

Error is notified and notified the abnormality exception if issuing this API on a portion other than task portion or quasi-task portion.

Error is notified if issuing this API on the self domain with dispatch disabled. TRON Safe Kernel operates as follows.

(1) Terminate the task issuing this API

(2) Release from dispatch disabled

(3) Notify an abnormal exception

8.2.1.5 ts_exd_tsk/tn_exd_tsk – Exit and Delete Task

C Language Interface

[Safety API] void ts_exd_tsk(void);

[Normal API] void tn_exd_tsk(void);

Parameter

Nome

Return parameter

None (NOT returned to the context issuing API)

Error code

None (Notify by an abnormality exception if an error detected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API terminates the invoking task and deletes. This API transits the task to non-existent state (NON-EXISTENT).

This API is NOT returned to the context at issuing even if detecting any errors. Therefore, error code cannot be returned directly as a return parameter of API. This API notifies an abnormality exception if detecting any errors.

Error 'E_CTX' is notified and notified the abnormality exception if issuing this API on a portion other than task portion or quasi-task portion.

Error is notified if issuing this API on the self domain with dispatch disabled. TRON Safe Kernel operates as follows.

(1) Terminate the task issuing this API

(2) Release from dispatch disabled

(3) Notify an abnormal exception

8.2.1.6 ts_ter_tsk/tn_ter_tsk – Terminate Task

C Language Interface

[Safety API]	ER ercd = ts_ter_tsk(ID tskid);
[Normal API]	ER ercd = tn_ter_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number (‘tskid’ is invalid or cannot be used)
E_NOEXS	Object does not exist (the task specified in ‘tskid’ does not exist)
E_OBJ	Object state Invalid (Task NOT on dormant state (DORMANT) or Invoking task)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API terminates the task set by 'tskid' forcibly.
Its task is transited to dormant state (DORMANT).

Information for task such as task priority, etc. are set to the initial values at the task returned to on dormant state (DORMANT).
For example, if a task with priority changed by 'ts_chg_pri'/'tn_chg_pri' is terminated, its priority is resumed to the task start-up priority ('itskpri') set by 'ts_cre_tsk'/'tn_cre_tsk'. Therefore, if the task started again by 'ts_sta_tsk'/'tn_sta_tsk', its priority is different from the one at executing this API in some cases.

The task is, if on waiting state (including on suspended state), released from its state and terminated. Its task is deleted from on any queue (such as a semaphore queue and so on) if on queuing.

It is NOT possible to set the invoking task to this API. Error 'E_OBJ' is notified if setting.

Relations between task state and operation result by this API are shown in ‘Table 8-1 Relation between execution of target

task and result of execution'.

Table 8-1 Relation between execution of target task and result of execution

Task state	Return code	Operation
Running and Ready state (RUNNING, READY) (other than the invoking task)	E_OK	Exit forcibly
Running state (RUNNING) (the invoking task)	E_OBJ	No operation
Waiting state (WAITING)	E_OK	Exit forcibly
Suspended state (SUSPENDED)	E_OK	Exit forcibly
Waiting and suspended state (WAITING-SUSPENDED)	E_OK	Exit forcibly
Dormant state (DORMANT)	E_OBJ	No operation
Non-existent state (NON-EXISTENT)	E_NOEXS	No operation

8.2.1.7 ts_chg_pri/tn_chg_pri – Change Task Priority

C Language Interface

[Safety API]	ER ercd = ts_chg_pri(ID tskid, PRI tskpri);
[Normal API]	ER ercd = tn_chg_pri(ID tskid, PRI tskpri);

Parameter

ID	tskid	Task ID
PRI	tskpri	Priority of task

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the task specified in 'tskid' does not exist)
E_PAR	Parameter error ('tskpri' is invalid or NOT used)
E_ILUSE	Use illegally (Upper limit priority is Violated)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API changes the base priority of task set by 'tskid' to the priority set by 'tskpri'. This API also changes current priority in the same way.

'tskpri' : It is possible to set the value between '1' and 250, lower value has higher priority.

The priority which can be specified is set on the configuration statically for each domain..

And 'TPRI_INI' (=0) can be specified as task priority regardless of the configuration.

'tskid' : The task is itself if setting TSK_SELF(=0).

This API changes the base priority of the task to task start-up priority ('itskpri)' if setting TPRI_INI(=0) to 'tskpri'.

Priority changed by this API is effective until the task finished. Transiting the task to dormant state (DORMANT), current task priority is discarded and then changed to task start-up priority, set at creating, 'itskpri'.

Priority set on dormant state is still effective. The task has its priority if starting at the next time.

If current target priority is same as the base one after executing this API (always met in case of not used Mutex functions), this API operates as follows.

- If the task being on running and ready state, this API changes task priority order on the basis of new priority. The task has the lowest priority order among ones with same priority as new.
- If the task being on any queue, this API changes task priority order on the basis of new priority. The task has the end of queue among ones with same priority as new.
- Error 'E_ILUSE' is notified if the task locking the Mutex with 'TA_CEILING' or waiting for lock, and if base priority set by 'tskpri' being higher than the upper limit priority of any Mutex.

In case of task priority order in the queue changing as a result by this API, the task or others on the queue is released from waiting state in some cases (For example, queue for message buffer in sending).

In case of the task being on waiting state for locking Mutex with 'TA_INHERIT', this API executes transitive priority inheritance operation in some cases, because of changing the base priority.

If not using Mutex functions, in case of setting the invoking task to the task, base priority to new priority and calling this API, execution order of the task has the lowest priority among the ones with same priority.

8.2.1.8 ts_inf_tsk/tn_inf_tsk – Reference Task Statistics

C Language Interface

[Safety API]	ER ercd = ts_inf_tsk(ID tskid, T_ITSK *pk_itsk, BOOL clr);
[Normal API]	ER ercd = tn_inf_tsk(ID tskid, T_ITSK *pk_itsk, BOOL clr);

Parameter

ID	tskid	Task ID
T_ITSK*	pk_itsk	Pointer for the area returning statistical information for task
BOOL	clr	Clear statistical information for task or not (valid in Safety API only)

Return parameter

ER	ercd	Error code
T_ITSK*	itsk	Statistical information for task

Statistical information for task

```
typedef struct st_itsk {
        RELTIM    ctime;        /* Accumulated task execution time ( by milliseconds) */
        RELTIM    stime;        /* Consecutive task execution time (by milliseconds) */
} T_ITSK;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number (‘tskid’ is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_PAR	Parameter error (Address set by 'pk_itsk' is invalid)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to ‘pk_itsk’. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers to statistical information for the task set by 'tskid'.

It is possible to set the invoking task by tskid=TSK_SELF(=0).

'ctime' means accumulated task execution time,.accumulated time on executing by milliseconds. The time is started to

measure at the task executed by 'ts_sta_tsk/tn_sta_tsk'. It is added the executing time of the task (including executing time of an extended SVC called by the task), not added the execution time of other execution unit (such as interrupt handler and so on).

'ctime' is reset (clear to '0') in the case as follows.

- Start to execute the task by 'ts_sta_tsk/tn_sta_tsk'
- Execute on setting 'clr=TRUE' to 'ts_inf_tsk' (Reset after getting a value)

'stime' means consecutive task execution time.

'stime' means consecutive task execution time, measured on executing consecutively by milliseconds. Its time corresponds to the interval from starting the program for the task to changing to the state except for 'RUNNING' state.. Any interrupt has no effect. It is added the executing time of the task (including executing time of an extended SVC called by the task), not added the execution time of other execution unit (such as interrupt handler and so on).

- 'stime' is reset (clear to '0') in the following case.

Execution task is dispatched and transits to the state except for 'RUUNING' state.

In Safety API, if 'clr=TRUE', accumulated task execution time is reset (clear to '0') after getting statistical information.

In Normal API, setting of 'clr' is ignored.

8.2.1.9 ts_ref_tsk/tn_ref_tsk – Reference Task Status

C Language Interface

[Safety API] ER ercd = ts_ref_tsk(ID tskid, T_RTSK *pk_rtsk);

[Normal API] ER ercd = tn_ref_tsk(ID tskid, T_RTSK *pk_rtsk);

Parameter

ID	tskid	Task ID
T_RTSK*	pk_rtsk	Pointer for the area returning task state

Return parameter

ER	ercd	Error code
T_RTSK	rtsk	Task state

Task state

```
typedef   struct st_rtsk {
          ID        dmnno;          /* Domain number */
          ATR       tskatr;         /* Task attribute */FP
          FP        task;           /* Task start-up address */
          RELTIM    maxrtim;        /* Maximum consecutive execution time */
          INT       stksz;          /* Use stack size (numner of bytes) */
          INT       sstksz;         /* System stack size (numner of bytes) */
          PRI       itskpri;        /* Start-up priority */
          PRI       tskpri;         /* Current priority */
          PRI       tskbpri;        /* Base priority */
          UINT      tskstat;        /* Task state */
          UINT      tskwait;        /* Waiting factor */
          ID        wid;            /* Waiting object ID */
          INT       wupcnt;         /* Number of wakeup requests */
          INT       suscnt;         /* Number of nests for suspended */
          UINT      waitmask;       /* Waiting factor for disabling of wait */
          UB        oname[8];       /* Object name */
} T_RTSK;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object not existed (Task set by 'tskid' is NOT existed)
E_PAR	Parameter illegal (Address set by 'pk_rtsk' is invalid)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is

	protected)
E_MACV	Memory access violation (There is no access authority to 'pk_rtsk'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers to the state of the task set by 'tskid'.

It is possible to set the invoking task by 'tskid=TSK_SELF(=0)'. Error E_ID is notified if setting 'tskid=TSK_SELF' on task independent part and time event handler.

'dmnno' means a number of the self domain.
'tskpri' means current priority for the task.
'tskbpri' means base priority for the task.

'tskstat' means task state.
Task states are indicated as follows.

TTS_RUN	0x00000001	Running state (RUNNING)
TTS_RDY	0x00000002	Ready state (READY)
TTS_WAI	0x00000004	Waiting state (WAITING)
TTS_SUS	0x00000008	Suspended state (SUSPENDED)
TTS_WAS	0x0000000c	Waiting and suspended state (WAITING-SUSPENDED)
TTS_DMT	0x00000010	Dormant state (DORMANT)
TTS_NODISWAI	0x00000080	State of disabling of wait

Task states by 'TTS_RUN', 'TTS_WAI', etc. are set by bits.
'TTS_NODISWAI' is set to 'TTS_WAI' if the task being on a state disabling of wait by 'ts_dis_wai/tn_dis_wai'.'TTS_NODISWAI' is not set to other than 'TTS_WAI'.
This API returns running state ('TTS_RUN') as 'tskstat' if executing 'ts_ref_tsk' for the interrupted task by interrupt handler.

'tskwait' means a waiting factor of the task.
'wid' means an object ID of the task on waiting state.
'tskwait' and 'wid' are set '0' if the task not being on waiting state.
In case of the task being on waiting state, that is, 'tskstat' set 'TTS_WAI', values of 'tskwait' and 'wid' are shown in 'Table 8-2 Value of 'tskwait' and 'wid''.

Table 8-2 Value of 'tskwait' and 'wid'

tskwait	Value	Description	wid
TTW_SLP	0x00000001	Waiting by 'ts_slp_tsk	0
TTW_DLY	0x00000002	Waiting by 'ts_dly_tsk	0
TTW_SEM	0x00000004	Waiting by 'ts_wai_sem'	semid'semid' of waiting target
TTW_FLG	0x00000008	Waiting by 'ts_wai_flg	'flgid' of waiting target
TTW_MTX	0x00000080	Waiting by 'ts_loc_mtx	'mtxid' of waiting target
TTW_SMBF	0x00000100	Waiting by 'ts_snd_mbf'	'mbfid' of waiting target
TTW_RMBF	0x00000200	Waiting by 'ts_rcv_mbf'	'mbfid' of waiting target

'wupcnt' means number of wakeup requests.

Number of wakeup requests is the count of issuing 'ts_wup_tsk/tn_wup_tsk' for the task on other than sleep state.

'waitmask' means a wating factor of disabliing of wait.

It is the waiting factor that the task is prohibited transiting to waiting state, that is, the waiting factor that is set the task by parameters on 'ts_dis_wai/tn_dis_wai'. 'waitmask' has the same bits as one of 'tskwait'.

Task on dormant state (DORMANT) has 'wupcnt=0'.

8.2.1.10 ts_slp_tsk/tn_slp_tsk – Sleep Task

C Language Interface

[Safety API]	ER ercd = ts_slp_tsk(TMO tmout);
[Normal API]	ER ercd = tn_slp_tsk(TMO tmout);

Parameter

TMO	tmout	Timeout setting (by milliseconds)

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error ('tmout' is invalid)
E_RLWAI	Release from waiting state forcibly (Enable to set 'ts_rel_wai/tn_rel_wai' on waiting)
E_DISWAI	Release from waiting by disabling of wait
E_TMOUT	Fail a polling or Timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API transits the invoking task from running state (RUNNING) to sleep state (waiting for setting 'ts_wup_tsk/tn_wup_tsk'). If queuing the wakeup request for the invoking task, that is, if number of wakeup requests is '1' and over, this API continues to execute, subtracting '1' on the wakeup request and not transiting the invoking task to waiting state.
'tmout' : This API exits normally if 'ts_wup_tsk/tn_wup_tsk' set for the task issuing this API before passing the time set by 'tmout'.
Timeout error 'E_TMOUT' is notified if not set 'ts_wup_tsk/tn_wup_tsk' before passing the time set by 'tmout'.
Timeout is infinite if setting TMO_FEVR(=-1) to 'tmout'. Tthis API waits until issuing 'ts_wup_tsk'. Domain error 'E_DOMAIN' is notified if 'TMO FEVER' is set for Safety API..

This API exits, not transiting to waiting state, if setting TMO_POL(=0) to 'tmout'.
Error 'E_TMOUT' (Fail on polling) is notified if number of wakeup requests set '0'. This API subtracts '1' on the wakeup request and exits normally if number of wakeup requests set '1' and over.
If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the

value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

8.2.1.11 ts_wup_tsk/tn_wup_tsk – Wakeup Task

C Language Interface

[Safety API]	ER ercd = ts_wup_tsk(ID tskid);
[Normal API]	ER ercd = tn_wup_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (Task is NOT on dormant state (DORMANT))
E_QOVR	Overflow a queuing or nesting (Overflow a number of queuings 'wupcnt')
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API releases from waiting state if the task set by 'tskid' is on sleep state.

It is NOT possible to set the invoking task. Error 'E_OBJ' is notified if setting.

Wakeup request by this API is queued if the task not being on waiting state by executing 'ts_slp_tsk/tn_slp_tsk'. Issuing of this API for the task is recorded, and then its task continues to execute, not transited to waiting state, if executing 'ts_slp_tsk/tn_slp_tsk' by the task. This operation is called 'Queuing of wakeup request'.
Operation for the queuing of wakeup request is described in details as follows.

Each task has information for number of wakeup requests ('wupcnt') and its initial value (one on executing 'ts_sta_tsk/tn_sta_tsk') is '0'. Number of wakeup requests for the task is incremented by '1' by executing this API for the task on other than sleep state. It is decreased by '1' by executing 'ts_slp_tsk/tn_slp_tsk'. And then, if the task with number of wakeup requests '0' executes 'ts_slp_tsk/tn_slp_tsk', its task transits to waiting state, not be negative.
Maximum number of wakeup requests is set to 'TS_MAX_WUPCNT', defined on the configuration statically.

Error 'E_QOVR' is notified if issuing this API over the maximum number of wakeup requests.

8.2.1.12 ts_can_wup/tn_can_wup – Cancel Wakeup Task

C Language Interface

[Safety API]	INT wupcnt = ts_can_wup(ID tskid);
[Normal API]	INT wupcnt = tn_can_wup(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

INT	wupcnt	Number of wakeup requests for queue
	or	Error code

Error code

E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (The task is NOT on dormant state (DORMANT))
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API cancels all wakeup requests for the task set by 'tskid'.

Number of wakeup requests for the task is set '0'.

Return value at exiting normally is the number of wakeup requests at issuing this API.

It is possible to set the invoking task by 'tskid=TSK_SELF(=0)'.

8.2.1.13 ts_rel_wai/tn_rel_wai – Release Wait

C Language Interface

[Safety API]	ER ercd = ts_rel_wai(ID tskid);
[Normal API]	ER ercd = tn_rel_wai(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (The task is NOT on waiting state (including the invoking task and on dormant state (DORMANT))
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API releases the task set by 'tskid' from on any waiting state (excluding on suspended state (SUSPENDED)) forcibly. Error 'E_RLWAI' is notified if releasing from on waiting state by this API. The task is always released from waiting state without reserving a resource (not meeting release conditions).

Queuing of requests for releasing from waiting state is not operated on this API. The task set by 'tskid' is released if on waiting state, Error 'E_OBJ' is notified if not. Error 'E_OBJ' is also notified if setting the invoking task.
Release from suspended state (SUSPENDED) is not operated on this API. Error 'E_OBJ' is notified if setting the task on suspended state.

This API transits the task from waiting and suspended state (WAITNG-SUSPENDED) to on suspended state (SUSPENDED).
Relations between task state and execution result on 'ts_rel_wai/tn_rel_wai' are shown in 'Table 8-3 Relations between task state and execution result'.

Table 8-3 Relations between task state and execution result

State of the task	Return code	Operation
Running and ready state (RUNNING,READY) (other than the invoking task)	E_OBJ	No operation
Running state (RUNNING) (invoking task)	E_OBJ	No operation
Waiting state (WAITNG)	E_OK	Release from waiting state
Suspended state (SUSPENDED)	E_OBJ	No operation
Waiting and suspended state (WAITING-SUSPENDED)	E_OK	Transit to suspended state
Dormant state (DORMANT)	E_OBJ	No operation
Non-existent state (NON=EXISTENT)	E_NOEXS	No operation

8.2.1.14 ts_sus_tsk/tn_sus_tsk – Suspend Task

C Language Interface

[Safety API] ER ercd = ts_sus_tsk(ID tskid);

[Normal API] ER ercd = tn_sus_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (Task is the invoking task or on dormant state (DORMANT))
E_CTX	Set to the task on running state in dispatch disabled state
E_QOVR	Overflow a queue or nest (Overflow number of nests 'suscnt')
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API transits the task set by 'tskid' to suspended state (SUSPENDED) and suspends its execution.

Suspended state (SUSPENDED) is released by issuing 'ts_rsm_tsk/tn_rsm_tsk' and 'ts_frsm_tsk/tn_frsm_tsk'.

If the task on waiting state, its task is transited by this API to waiting and suspended state (WAITING-SUSPENDED), composed of waiting state and suspended state. And then, the task is transited to suspended state (SUSPENDED) at meeting conditions for releasing waiting. The task is resumed to previous waiting state if issuing 'ts_rsm_tsk/tn_rsm_tsk'.

It is NOT possible to set transiting to suspended state (SUSPENDED) to the invoking task, because of suspended state being equivalent to break one by API issued by other task. Error 'E_OBJ' is notified if setting the invoking task.

Error 'E_CTX' is notified if setting the task on running state (RUNNING) under dispatch disabled in case of issuing this API on task independent part and time event handler.

If issuing this API repeatedly, the task is transited to multiple suspended states (SUSPENDED). This is called 'Nest of suspended state request'. The task is resumed by executing 'ts_rsm_tsk/tn_rsm_tsk' in number of the API issuings ('suscnt'). It is possible to nest a pair of 'ts_sus_tsk – ts_rsm_tsk / tn_sus_tsk – tn_rsm_tsk'.

Maximum number of nests for suspended request is set to 'TS_MAX_SUSCNT', defined on the configuration statically.
Error 'E_QOVR' is notified if issuing 'ts_sus_tsk/tn_sus_tsk' over the maximum number of nests for suspended request.

[Supplement]
Resource (such as semaphore and so on) is assigned on the same condition of other than suspended state (SUSPENDED), even if the task on waiting state for getting a resource and on suspended state (SUPENDED). The condition and priority for assigning a resource and releasing from on waiting state is the same as on other state, not delayed to assign a resource, even if on suspended state (SUSPENDED). Suspended state (SUSPENDED) is the orthogonal relation with the other operations and task states.

8.2.1.15 ts_rsm_tsk/tn_rsm_tsk – Resume Task

C Language Interface

[Safety API] ER ercd = ts_rsm_tsk(ID tskid);

[Normal API] ER ercd = tn_rsm_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (The task is NOT on suspended state (including the invoking task and on dormant state (DORMANT))
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API releases the task set by 'tskid' from on suspended state (SUSPENDED).
This API releases the task from suspended state (SUSPENDED) by 'ts_sus_tsk/tn_sus_tsk' and resumes to the execution.

The task is released from suspended state (SUSPENDED) only and transited to waiting state (WAITING), if on waiting and suspended state (WAITING-SUSPENDED). (Refer to 'Figure 3-1 State Transition of Task'.)

It is NOT possible to set the invoking task. Error 'E_OBJ' is notified if setting.

This API releases one (1) nest of suspended request ('suscnt'). If issuing 'ts_sus_tsk/tn_sus_tsk' in multiple ('suscnt >= 2'), the task is on suspended state (SUSPENDED) after this API executed.

If the task on running state (RUNNING) or ready state (READY) transited to suspended state (SUSPENDED) and resumed the execution by 'ts_sus_tsk/tn_sus_tsk', its task has the lowest priority among those with same priority.
For example, if 'task_A' and 'task_B' having same priority, this API is operated as follows.

```
ts_sta_tsk ( tskid=task_A, stacd_A );
ts_sta_tsk ( tskid=task_B, stacd_B );
/* 'task_A' is prior to 'task_B' because of FCFS(First Come First Served) criteria. */

ts_sus_tsk ( tskid=task_A );
ts_rsm_tsk ( tskid=task_A );
/* 'task_B' is prior to 'task_A'. */
```

8.2.1.16 ts_frsm_tsk/tn_frsm_tsk – Force Resume Task

C Language Interface

[Safety API]	ER ercd = ts_frsm_tsk(ID tskid);
[Normal API]	ER ercd = tn_frsm_tsk(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_OBJ	Object state Invalid (The task is NOT on suspended state (SUSPENDED) (including the invoking task and on dormant state (DORMANT))
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API releases the task set by 'tskid' from all suspended state (SUSPENDED).
This API releases the task from all suspended state (SUSPENDED) by 'ts_sus_tsk/tn_sus_tsk' and resumes to the execution.

The task is released from suspended state (SUSPENDED) only and transited to waiting state (WAITING), if on waiting and suspended state (WAITING-SUSPENDED).

It is NOT possible to set the invoking task. Error 'E_OBJ' is notified if setting.

This API releases all nests of suspended request (suscnt=0), that is, all requests are released (suscnt=0) if issuing 'ts_sus_tsk/tn_sus_tsk' in multiple ('suscnt' >= 2). Therefore, the task is always released from suspended state (SUSPENDED) and then enable to resume to execute on the state.
This API releases all nests of suspended request ('suscnt'). If issuing 'ts_sus_tsk/tn_sus_tsk' in multiple ('suscnt >= 2'), all requests are released ('suscnt=0'). The task is always released from suspended state (SUSPENDED). It is possible to resume the task on other than waiting and suspended state (WAITING-SUSPENDED).

If the task on running state (RUNNING) or ready state (READY) transited to suspended state (SUSPENDED) and resumed the execution by 'ts_sus_tsk/tn_sus_tsk', its task has the lowest priority among those with same priority.
For example, if 'task_A' and 'task_B' having same priority, this API is operated as follows.

```
ts_sta_tsk ( tskid=task_A, stacd_A );
ts_sta_tsk ( tskid=task_B, stacd_B );
/* 'task_A' is prior to 'task_B' because of FCFS criteria. */

ts_sus_tsk ( tskid=task_A );
ts_frsm_tsk ( tskid=task_A );
/* 'task_B' is prior to 'task_A'. */
```

8.2.1.17 ts_dly_tsk/tn_dly_tsk – Delay Task

C Language Interface

[Safety API]	ER ercd = ts_dly_tsk(RELTIM dlytim);
[Normal API]	ER ercd = tn_dly_tsk(RELTIM dlytim);

Parameter

RELTIM	dlytim	Delay time (by milliseconds)

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion , or issued in dispatch disabled state or in interrupt disabled state)
E_RLWAI	Release from suspended state (Receive 'ts_rel_wai/tn_rel_wai' in waiting)
E_DISWAI	Release a waiting by disabling of wait

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API breaks the execution of the invoking task and transits its task to the state of waiting for passing the time
Break time for the task is set by 'dlytim'.

State of waiting for passing the time is one of waiting state. It is possible to cancel the waiting by 'ts_rel_wai/tn_rel_wai'.
The time counts even if the task on waiting and suspended state (WAITING-SUSPENDED).

[Supplement]

This API exits normally if passing the time set as delay time, not same as 'ts_slp_tsk/tn_slp_tsk'. The task is not released from waiting state even if executing 'ts_wup_tsk/tn_wup_tsk' in delay time. This API exits in delay time only if issuing 'ts_rel_wai/tn_rel_wai'., ts_dis_wai/tn_dis_wai and ts_ter_tsk/tn_ter_tsk.

8.2.1.18 ts_dis_wai/tn_dis_wai – Disable Task Wait

C Language Interface

[Safety API] INT tskwait = ts_dis_wai(ID tskid, UINT waitmask);

[Normal API] INT tskwait = tn_dis_wai(ID tskid, UINT waitmask);

Parameter

ID	tskid	Task ID
UINT	waitmask	Setting to disable task wait

Return parameter

INT	tskwait	Task waiting state after disabling task wait
	or	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' NOT existed)
E_PAR	Parameter Invalid ('waitmask' Invalid)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task dependent part portion
○	○	○

Description

This API prohibits the task set by 'tskid' from waiting state on the basis of waiting factor set by 'waitmask'.

The task is released from waiting state if already on waiting state with waiting factor set by 'waitmask'.

'waitmask' : Set by the logical add (OR) combined the waiting factor as described below.

```
#define TTW_SLP     (0x00000001U)   /* Wait for wakeup */
#define TTW_DLY     (0x00000002U)   /* Wait for task delay */
#define TTW_SEM     (0x00000004U)   /* Wait for semaphore */
#define TTW_FLG     (0x00000008U)   /* Wait for event flag */
#define TTW_MTX     (0x00000080U)   /* Wait for Mutex */
#define TTW_SMBF    (0x00000100U)   /* Wait for sending on message buffer */
#define TTW_RMBF    (0x00000200U)   /* Wait for receiving on message buffer */
#define TTX_SVC     (0x80000000U)   /* Prohibit calling an extended SVC  */
```

Error 'E_PAR' is notified if setting the bit other than described above or not setting any bits described above.

The task is transited to new state of disabling wait set by 'waitmask' if already on other state of disabling wait. All states of disabling wait are released by executing 'ts_ena_wai/tn_ens_wai'.

'TTX_SWC'is not a waiting state of task, but it is special setting prohibiting calling the extended SVC.
If trying to call an extended SVC by the task on 'TTX_SVC', this API returns error 'E_DISWAI', not calls an extended SVC. It is NOT possible to finish executing the extended SVC already called.

This API returns waiting state of the task after operating disabling wait as return value ('tskwait'), same as one of 'tskwait' on 'ts_ref_tsk/tn_ref_tsk", excluding information of 'TTX_SVC'.

The task is not transited to waiting state (or released from waiting state) if 'tskwait' set '0'. Its task is on waiting state by the waiting factor other than set by 'waitmask' if not.

Error 'E_DISWAI' is returned if releasing the task from on waiting state or trying to transit from state of disalbling wait to waiting state by this API.

Error 'E_DISWAI' is notified if calling the API possible to transit from state of disabling wait to waiting state, even if enabling to operate without the waiting. This is the same even if TMO_POL is specified.
For example, if sending to a message buffer ('ts_snd_mbf/tn_snd_mbf) with sufficient space, possible to send a message without waiting, Error 'E_DISWAI' is notified, not sent any messages.

If calling an extended SVC, disabling wait is canceled automatically and then resumed previous settings after returning from the extended SVC. Disabling wait set on executing the extended SVC is canceled at returning to the caller automatically.

Disabling wait is also canceled automatically if transiting the task to dormant state (DORMANT). Its disabling wait is effective on dormant state (DORMANT) and applied to the task at the next time.
Validity period of disabling wait is shown in 'Figure 8-1 Validity period of disabling wait'.

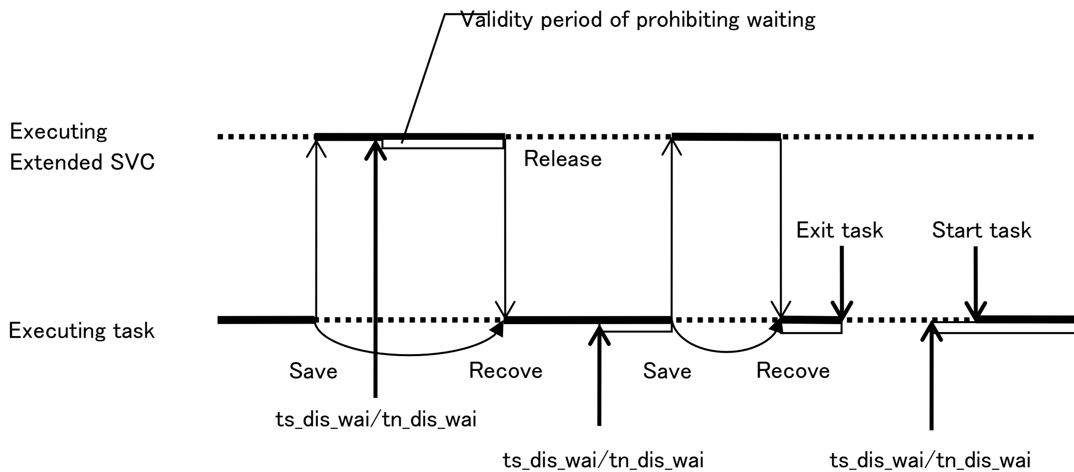

Figure 8-1 Validity period of disabling wait

It is possible for some objects such as semaphore, etc. to set 'TA_NODISWAI' to its object attributes at creating. Waiting is not disabled in the object created with 'TA_NODISWAI'.
'Waiting is not disabled' means if the task disabling wait on specific object by this API operates the object created with 'TA_NODISWAI', its object is executed normally by ignoring disabling wait. It does not mean an error is notified for a rejection of disabling wait at issuing this API.

Return value is not '0' even if setting the object to TA_NODISWAI attributes.
For example, if issued for the task waiting for the semaphore with 'TA_NODISWAI', its task is not released from waiting state for semaphore. Return value is 'TTW_SEM(0x00000004)'.

It is possible to set the invoking task by 'tskid=TSK_SELF(=0)'.

8.2.1.19 ts_ena_wai/tn_ena_wai – Enable Task Wait

C Language Interface

[Safety API]	ER ercd = ts_ena_wai(ID tskid);
[Normal API]	ER ercd = tn_ena_wai(ID tskid);

Parameter

ID	tskid	Task ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('tskid' is invalid or cannot be used)
E_NOEXS	Object NOT existed (Task set by 'tskid' is NOT existed)
E_DACV	Domain access protection violation (Target task belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event hander	Task independent portion
○	○	○

Description

This API releases ALL disabling wait settings for the task set by 'tskid'.

It is possible to set the invoking task by 'tskid=TSK_SELF(=0).

8.2.2 API for System State Management Functions

Refer to ‘3.2 System State Management Functions’ about the detail of System State Management Functions.

8.2.2.1 ts_rot_rdq/tn_rot_rdq – Rotate Ready Queue

C Language Interface

[Safety API]	ER ercd = ts_rot_rdq(PRI tskpri);
[Normal API]	ER ercd = tn_rot_rdq(PRI tskpri);

Parameter

PRI	tskpri	Task priority

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error ('tskpri' is invalid)
E_CTX	Context error

Valid Context

Task portion	Time event handler	Task independent portion
○	○	×

Description

This API rotates the order of task priority set by 'tskpri' in the self domain.

The task with the highest priority in the task whose priority is 'tskpri' in the self domain is set the lowest priority among tasks with same priority.

It is possible for task priority 'tskpri' to set the value between '1' to '250', lower value higher priority.

Actually specified priority is defined by the configuration statically for each domain.

Error 'E_PAR' is notified if setting higher priority than the maximum priority of the self domain.

If 'tskpri=TPRI_RUN(=0)', this API rotates the task priority order for the task on running state (RUNNING) in the self domain at issuing.

On 'ts_rot_rdq/tn_rot_req' issued by general task, this operation is equivalent to a rotation of the task priority order for the task with the priority same as that of the invoking task. Additionally, it is available to issue 'ts_rot_rdq/tn_rot_rdq (tskpri=TPRI_RUN') on time event handler.

If the task on running and ready state with the priority is only one, the priority order is not changed as a result. Consecutive execution time of task is cleared.

This API has no operation (error not occurred) if no task on running and ready state with the priority.

[Supplement]

Execution order of the invoking task has the lowest one among the task with same priority if setting 'TPRI_RUN' or current priority of the invoking task to the priority on permitting dispatching. It enables to release execution permission.
On dispatch disabled state, the execution order of the invoking task is not always the lowest one among the task with same priority because of not always executing the task with the highest priority among the task with same priority.

8.2.2.2 ts_get_tid/tn_get_tid –Get Task Identifier

C Language Interface

[Safety API]	ID tskid = ts_get_tid(void);
[Normal API]	ID tskid = tn_get_tid(void);

Parameter

None

Return parameter

ID	tskid	ID of task on running state

Error code

None

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API gets Task ID of the task on running state on the self domain currently.

It returns '0' if no task on running state in the self domain.

Task on running state currently, except for executing a time event handler, is the invoking task. Task ID of the task on running state currently in the self domain is returned also if issuing on time event handler.

[Supplement]

Task ID returned by this API is the same as 'runtskid' returned by 'ts_ref_sys/tn_ref_sys'.

8.2.2.3 ts_dis_dsp/tn_dis_dsp – Disable Dispatch

C Language Interface

[Safety API]	ER ercd = ts_dis_dsp(TMO tmout);
[Normal API]	ER ercd = tn_dis_dsp(TMO tmout);

Parameter

TMO	tmout	Time of dispatch disabled

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error (‘tmout’ is invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API prohibits dispatching the task on the self domain.

Time of dispatch disabled is specified to ‘tmout’. If ‘TMO FEVER(=1)’ is specified as ‘tmout’ it shows that time of dispatch disabled is infinite. However if ‘TMO FEVER’ is specified in Safety domain , domain error ‘E_DOMAIN’ is notified. And if ‘TMO POL(=0)’ is specified in this domain , parameter error ‘E_PAR’ is notified.
If smaller than or equal to ‘−2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

After this API is issued ,until the time of dispatch disabled elapses or the state of dispatch disabled is released by 'ts_ena_dsp/tn_ena_dsp', the self domain becomes the state of dispatch disabled and the invoking task is not transited from running state (RUNNING) to ready state (READY) and to other than running state such as waiting state, etc..

This API operates on the state of dispatch disabled as follows.

- If the task with higher priority than that of the task executing this API transited to ready state (READY), dispatching to its task is not operated. Dispatching to the task with higher priority is delayed until finishing on the state of dispatch disabled.
- Error 'E_CTX' is notified if the task executing this API issues a system call possible to transit the task to waiting state

such as 'ts_slp_tsk/tn_slp_tsk', 'ts_wai_sem/tn_wai_sem' and so on.

- 'TSS_DDSP' is returned as 'sysstat' if referring system state by 'ts_ref_sys/tn_ref_sys'.

It is NOT possible for the task to transit from running state (RUNING) to on dormant state (DORMANT) or non-existent state (NON-EXISTENT) on state of dispatch disabled. Error 'E_CTX' is notified if issuing 'ts_ext_tsk/tn_ext_tsk' or 'ts_exd_tsk/tn_exd_tsk' on state of dispatch disabled. This is operated as follows.

- Finish the task issuing 'ts_ext_tsk/tn_ext_tsk' (NOT return to caller).
- Finish the task issuing 'ts_exd_tsk/tn_exd_tsk' and Delete (NOT return to caller).
- Release from state of dispatch disabled automatically.
- Notify an abnormality exception.

If issuing this API in the self domain already on state of dispatch disabled, the state of dispatch disabled continues, error not notified. Time of dispatch prohibition is overwritten and updated by 'tmout' at the timing when 'ts_dis_dsp/tn_dis_dsp' is issued again, And its state of dispatch disabled is released by one (1) issue of 'ts_ena_dsp/tn_ena_dsp' even if issuing this API in multiple. It is necessary to control its operation on nesting of the pair 'ts_dis_dsp/tn_dis_dsp – ts_ena_dsp/tn_ena_dsp' by user if used.

If not released from state of dispatch disabled after passing the time set by 'tmout', this API operates as follows.

- Release from state of dispatch disabled automatically.
- Notify an abnormality exception.

[Supplement]

Cyclic handler and Alarm handler are not prohibited executing on state of dispatch disabled. Cyclic handler and alarm handler are operated even if on the domain with dispatch disabled.
Dispatch disabled is valid in the self domain only. Task with higher priority on other domain than that of the task on the self domain is executed.

8.2.2.4 ts_ena_dsp/tn_ena_dsp – Enable Dispatch

C Language Interface

[Safety API] ER ercd = ts_ena_dsp(void);

[Normal API] ER ercd = tn_ena_dsp(void);

Parameter

None

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API permits dispatching to the task on the self domain. It is released from state of dispatch disabled on the self domain set by 'ts_dis_dsp/tn_dis_dsp'.

If the task on the self domain not prohibited dispatching issuing this API, the state of permitting dispatching is continued, error not notified.

8.2.2.5 ts_ref_sys/tn_ref_sys – Reference System Status

C Language Interface

[Safety API]	ER ercd = ts_ref_sys(T_RSYS *pk_rsys);
[Normal API]	ER ercd = tn_ref_sys(T_RSYS *pk_rsys);

Parameter

T_RSYS*	pk_rsys	Pointer of the area for returning information of system state

Return parameter

ER	ercd	Error code
T_RSYS	rsys	System state

System state

```
typedef struct st_rsys {
        UINT    sysstat;        /* System state */
        ID      runtskid;       /* ID of task on running state currently */
        ID      schedtskid;     /* ID of task next on running state */
} T_RSYS;
```

Error code

E_OK	Normal completion
E_PAR	Parameter error (Address set by 'pk_rsys' is invalid
E_MACV	Memory access violation (There is no access authority to 'pkrsys'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers to system state and returns the information as return parameter.

'**sysstat**' : means system state at calling this API.

Value of 'sysstat' is indicated as follows.

```
sysstat := ( TSS_TSK | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_QTSK | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_TEH | [TSS_DDSP] | [TSS_DINT] )
        || ( TSS_INDP )
```

TSS_TSK	On running the task portion

TSS_DDSP	On dispatch disabled
TSS_DINT	On prohibiting interrupting
TSS_INDP	On running a task independent part
TSS_QTSK	On running a quasi-task portion
TSS_TEH	On running a time event handler part

System state is defined in header file as follows.

```
#define TSS_TSK     (0x00000000U)   /* During execution of task part(context) */
#define TSS_DDSP    (0x00000001U)   /* During dispatch disable */
#define TSS_DINT    (0x00000002U)   /* During Interrupt disable */
#define TSS_INDP    (0x00000004U)   /* During execution of task independent part */
#define TSS_QTSK    (0x00000008U)   /* During execution of quasi-task part */
#define TSS_TEH     (0x00000010U)   /* During execution of time event handler */
```

'runtskid' : This API returns ID of task on running on the self domain currently to 'runtskid', ID of task next on running on the self domain to 'schedtskid'. This API returns '0' if no applicable task

'runtskid' is equal to 'schedtskid' normally. If wakeup the task with higher priority on state of dispatch disabled, 'runtskid' is not equal to 'schedtskid'.

It is possible to issue by an interrupt handler and a time event handler.

8.2.2.6 ts_ref_ver/tn_ref_ver – Reference Version Information

C Language Interface

[Safety API]	ER ercd = ts_ref_ver(T_RVER *pk_rver);
[Normal API]	ER ercd = tn_ref_ver(T_RVER *pk_rver);

Parameter

T_RVER* pk_rver	Pointer to the area for returning version information

Return parameter

ER	ercd	Error code
T_RVER	rver	Version information

Version information

```
typedef struct st_rver {
        UH      maker;          /* Maker code of kernel */
        UH      prid;           /* Identification number of kernel */
        UH      spver;          /* Version number of specifications */
        UH      prver;          /* Version number of kernel */
        UH      prno[4];        /* Control information for kernel products */
} T_RVER;
```

Error code

E_OK	Normal completion
E_PAR	Parameter error (Address set by 'pk_rver' is invalid
E_MACV	Memory access violation (There is no access authority to ‘pk_rver’. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

'pk_rver' : This API refers version information of using kernel and returns to the packet set by 'pk_rver'.

Refer to '3.2.2 Reference System ' in details.

8.2.3 API of the Synchronous/Communication Functions (Semaphore)

Refer to ‘3.3.1 Semaphore’ about the detail of the Synchronous/Communication Functions related to semaphore.

8.2.3.1 ts_cre_sem/tn_cre_sem – Create Semaphore

C Language Interface

[Safety API] ID semid = ts_cre_sem(CONST T_CSEM *pk_csem);

[Normal API] ID semid = tn_cre_sem(CONST T_CSEM *pk_csem);

Parameter

CONST T_CSEM*	pk_csem	Pointer to the semaphore creation information

Semaphore creation information

```
typedef   struct st_csem {
          ATR     sematr;          /* Semaphore attributes */
          INT     isemcnt;         /* Initial value of the semaphore count */
          INT     maxsem;          /* Maximum value of the semaphore count */
          UB      oname[8];        /* Object name */
} T_CSEM;
```

Return parameter

ID	semid	Semaphore ID
	or	Error code

Error code

E_LIMIT	A number of the semaphore over upper limit of the system
E_RSATR	Reserved attribute error ('sematr' is invalid or can NOT be used)
E_PAR	Parameter error (Address set by 'pk_csem' is invalid, or 'isemcnt' is negative or invalid, or maxsem is less than 0 or invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_ONAME	Object name error (Specified object name has been already used in the domain)
E_MACV	Memory access violation error (No access authority to 'pk_csem' or notarget memory area exists.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API creates a semaphore and assigns semaphore ID number.

'sematr' : a semaphore attribute is set for semaphore operation setting.

'sematr' is set as follows.

sematr := (TA_TFIFO || TA_TPRI) | (TA_FIRST || TA_CNT) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	Tasks are queued in FIFO order
TA_TPRI	Tasks are queued in priority order
TA_FIRST	The first task in the queue has precedence
TA_CNT	Tasks with fewer requests have precedence
TA_ONAME	Specifies an object name
TA_NODISWAI	Disabling of wait by ts_dis_wai/tn_dis_wai is prohibited

Semaphore attributes are defined in header file as follows.

```
#define TA_TFIFO        (0x00000000U)   /* manage queue by FIFO */
#define TA_TPRI         (0x00000001U)   /* manage queue by priority */
#define TA_FIRST        (0x00000000U)   /* first task in queue has precedence */
#define TA_CNT          (0x00000002U)   /* tasks with fewer requests have precedence */
#define TA_ONAME        (0x00000040U)   /* specifies an object name */
#define TA_NODISWAI     (0x00000080U)   /* reject request to disable wait */
```

The queuing order of tasks waiting for a semaphore is specified in TA_TFIFO or TA_TPRI.If the semaphore attribute is specified with TA_TFIFO, tasks are ordered by FIFO, and if TA_TPRI is specified, tasks are in order of their priority setting.

TA_FIRST and TA_CNT specify precedence of resource acquisition. TA_FIRST and TA_CNT do not change the order of the queue, which is determined by TA_TFIFO and TA_TPRI. notWhen TA_FIRST is specified, resources are allocated starting from the first task in the queue regardless of request resource. For this reason, as long as the first task in the queue cannot obtain the requested number of resources, tasks behind it in the queue are prevented from obtaining resources.not

TA_CNT means resources are assigned based on the order in which tasks are able to obtain the requested number of resources. The request resources are checked starting from the first task in the queue, and tasks to which their requested amount can be allocated receive resources.. If the resources are left after the assignment (the number of resources one (1) and over), the requested number of resources of tasks connected to the queue continues to check. Resource is not assigned by ascending order of the requested number of resources.

oname is valid when 'TA_ONAME' is set.

oname sets the object name of the target semaphore.

When TA_NODISWAI is specified, a disable wait by ts_dis_wai/tn_dis_wai is rejected

isemcnt specifies an initial value of semaphore count for the target semaphore. The value can be set between 0 and TS_MAX_SEMCNT.

maxsem specifies the maximum value of semaphore count for target semaphore (the maximum number of resources which the target semaphore can save).

The value to be specified to maxsem' is between '1' and 'TS_MAX_SEMCNT'.

TS_MAX_SEMCNTis decided statically by the configuration.

isemcnt shall be set a value of 'maxsem' and under. Error E_PAR is notified when the setting is over maxsem.

8.2.3.2 ts_del_sem/tn_del_sem – Delete Semaphore

C Language Interface

[Safety API]	ER ercd = ts_del_sem(ID semid);
[Normal API]	ER ercd = tn_del_sem(ID semid);

Parameter

ID	semid	Semaphore ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invaild ID number (semid is invalid or can NOT be used)
E_NOEXS	The Object does NOT existe (The semaphore of semid is NOT existe)
E_CTX	Context error (Issued in the portion other than atask portion and a quasi-task portion)
E_DACV	Domain access protection violation (The target semaphore belongs to other domain, and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

Delete specified semaphore by semid.

If there is a task waiting for condition fulfillment on the semaphore, Task waiting status is released, and error E_DLT is returned.

8.2.3.3 ts_sig_sem/tn_sig_sem – Signal Semaphore

C Language Interface

[Safety API]	ER ercd = ts_sig_sem(ID semid, INT cnt);
[Normal API]	ER ercd = tn_sig_sem(ID semid, INT cnt);

Parameter

ID	semid	Semaphore ID
INT	cnt	Number of returned resources

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number (semid is invalid or can NOT be used)
E_NOEXS	Object does not existe (The semaphore of semid does not existe)
E_QOVR	Queuing or nesting overflow (Overflow of the semaphore count)
E_PAR	Parameter error (cnt is '0' and under)
E_DACV	Domain access protection violation (The target semaphore belongs to other domain, and the access to it is proected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

Return resources to the specified semaphore by semid.cnt specifies the number of returning resources. cnt can set between '1' and TS_MAX_SEMCNT. TS_MAX_SEMCNT is decided statically by the configuration.
The semaphore count increses the number of cnt in the target semaphore.Error 'E_QOVR' isreturned when the semaphore count of the target semaphore is over the maximum value by returning resources in cntafter issuing this API. In this case, resources are not returned, the semaphore count of the tartget semaphore is not not changed.

When the count of the tartget semaphore is increased,, if the task waiting status for acquising semaphore in the tartget semaphore, check the requested number of resource,and assign resource if possible. The task assigned the resource is released from semaphore waiting status. Multiple tasks are assigned the resource and released from semaphore waiting status depends on conditions.

8.2.3.4 ts_wai_sem/tn_wai_sem – Wait on Semaphore

C Language Interface

[Safety API]	ER ercd = ts_wai_sem(ID semid, INT cnt, TMO tmout);
[Normal API]	ER ercd = tn_wai_sem(ID semid, INT cnt, TMO tmout);

Parameter

ID	semid	Semaphore ID
INT	cnt	Number of resource requests
TMO	tmout	Timout time set (by milliseconds)

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number(semid is invalid or is NOT used)
E_NOEXS	Object does NOT existe (The semaphore of semid does NOT existe)
E_PAR	Parameter error (cnt<=0, maxsem<cnt, 'tmout' is invaid)
E_DLT	Waiting object is deleted (The targer semaphore deleted due to wait)
E_RLWAI	Waiting status released forcibly (Set ts_rel_wai due to wait)
E_DISWAI	Wait released due to disabling of wait
E_TMOUT	Fail on polling or Timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issue in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation (The target semaphore belongs to other domain, and the access to it is protected)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

Acquire the cnt resources from the specified semaphore by semid.

cnt specifies the number of resources acquiring from the target semaphore. The specified value for cnt between 1 to maximum value of the semaphore count (maxsem).

Task issuing this API continues to execute without transiting to waiting status when resources can be acquired. In this case, the count of the target semaphore decreases cnt.

Task issuing this API is transited to semaphore waiting status when resource cannot be acquired. This means that the task is

connected to waiting queue for the target semaphore. In this case, the count of the target semaphore is not changed.

tmout specifies timeout time. Error E_TMOUT appears when the resources couldn't be acquired due to passing the specified timeout time.

Time to the timeout is infinite when TMO_FEVR(=−1) specified as tmout. In this case, waiting status has maintained untilresource can be acquired. Domain error E_DOMAIN appears when TMO_FEVR is specified in Safety API. Error E_TMOUT (polling error) appears without transiting to waiting status when 'TMO_POL(=0)' is specified as tmout.

If smaller than or equal to '−2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

8.2.3.5 ts_ref_sem/tn_ref_sem – Reference Semaphore Status

C Language Interface

[Safety API] ER ercd = ts_ref_sem(ID semid, T_RSEM *pk_rsem);

[Normal API] ER ercd = tn_ref_sem(ID semid, T_RSEM *pk_rsem);

Parameter

ID	semid	Semaphore ID
T_RSEM*	pk_rsem	Pointer to the area for returning a semaphore status

Return parameter

ER	ercd	Error code
T_RSEM	rsem	Semaphore status

State of semaphore

```
typedef     struct st_rsem {
            ID          dmnno;              /* domain number */
            ID          wtsk;               /* waiting Task ID */
            INT         semcnt;             /* current semaphore count value */
} T_RSEM;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number(semid is invalid or can NOT be used)
E_NOEXS	Object does NOT existe (The semaphore of semid does NOT existe)
E_PAR	Parameter error (Address set by 'pk_rsem' is invalid)
E_DACV	Domain access protection violation (The target semaphore belongs to other domains, and the access to it is protected).
E_MACV	Memory access violation error (No access authority to ‘pk_rsem’, or the target memory area doesn’t exist.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

Refer to status of semaphore set by semid,returns it as a return parameter.

dmnno means the domain number of the domain where it is belonged.

wtsk means ID of the task waiting status for this semaphore. If multible tasks are waiting, return the task ID to the head of queue. 'wtsk' is '0' when there is no waiting task.semcntmeans current value of semaphore count.

8.2.4 API for Synchronization and Communication Functions (Event Flag)

Refer to ‘3.3.2 Event Flag’ for details about Synchronization and Communication Functions of event flags.

8.2.4.1 ts_cre_flg/tn_cre_flg – Create Event Flag

C Language Interface

[Safety API] ID flgid = ts_cre_flg(CONST T_CFLG *pk_cflg);

[Normal API] ID flgid = tn_cre_flg(CONST T_CFLG *pk_cflg);

Parameter

CONST T_CFLG* pk_cflg Pointer to information for creating an event flag

Information for creating an event flag

```
typedef     struct st_cflg {
            ATR         flgatr;             /* Event flag attribute */
            UINT        iflgptn;            /* Event flag initial value */
            UB          oname[8];           /* Object name */
} T_CFLG;
```

Return parameter

ID	flgid	Event flag ID
	or	Error code

Error code

E_LIMIT	Number of event flags exceeds the system limit
E_RSATR	Reserved attribute error ('flgatr' is invalid or cannot be used)
E_PAR	Parameter error (Address specified in 'pk_cflg' is invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_ONAME	Object name error (Specified object name has already been used in the domain)
E_MACV	Memory access privilege error (There are no access privileges to 'pk_cflg' or there is no specified memory space.)

Valid Context

Task portion	Time event handler	Task-Independent portion
○	×	×

Description

Creates an event flag, assigning to it an event flag ID.

Sets the event flag attribute for specifying the operation of the specified event flag in flgatr.
flgatr is specified as follows.

flgatr := (TA_TFIFO || TA_TPRI) | (TA_WMUL || TA_WSGL) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	Tasks are queued in FIFO order
TA_TPRI	Tasks are queued in priority order
TA_WSGL	(Wait Single Task)
	Waiting by multiple tasks is not allowed (Wait Single Task)
TA_WMUL	Waiting by multiple tasks is allowed (Wait Multiple Tasks)
TA_ONAME	Specifies the object name
TA_NODISWAI	Disabling of wait by ts_dis_wai is prohibited

Event flag attributes are defined in the header file as shown below.

```
#define TA_TFIFO        (0x00000000U)   /* Manage task queue by FIFO */
#define TA_TPRI         (0x00000001U)   /* Manage task queue by priority */
#define TA_WSGL         (0x00000000U)   /* Waiting by multiple tasks is not allowed(Wait Single Task) */
#define TA_WMUL         (0x00000008U)   /* Waiting by multiple tasks is allowed(Wait Multiple Task) */
#define TA_ONAME        (0x00000040U)   /* Specifies the object name */
#define TA_NODISWAI     (0x00000080U)   /* Disabling of wait is prohibited */
```

When TA_WSGL is specified, multiple tasks cannot be in the WAITING state at the same time. Specifying TA_WMUL allows waiting by multiple tasks at the same time.

The queuing order of tasks waiting for a specified event flag are specified in TA_TFIFO or TA_TPRI.
If the event flag attribute is specified as TA_TFIFO, tasks are ordered by FIFO and TA_TPRI is specified, queuing of tasks gets ordered by their priority setting. When TA_WSGL is specified, however, since tasks cannot be queued, TA_TFIFO or TA_TPRI makes no difference.
When multiple tasks are waiting for an event flag, and event flag bit patterns are set, tasks are checked in order from the head of the queue, and the task event flag wait is released in order for tasks meeting the conditions. The first task to have its WAITING state released is therefore not necessarily the first in the queue.
If multiple tasks meet the conditions, wait state is released for each of them.
oname becomes valid if TA_ONAME is specified.
The object name of the specified event flag is set in oname
Disabling wait by ts_dis_wai/tn_dis_wai is prohibited when TA_NODISWAI is specified.

The initial value of the specified event flag bit pattern is set in iflgptn.

8.2.4.2 ts_del_flg/tn_del_flg – Delete Event Flag

C Language Interface

[Safety API]	ER ercd = ts_del_flg(ID flgid);
[Normal API]	ER ercd = tn_del_flg(ID flgid);

Parameter

ID	flgid	Event flag ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('flgid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the event flag specified in 'flgid' does not exist)
E_DACV	Domain access protection violation (Target event flag belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

Deletes the event flag specified in flgid.

Tasks in WAITING state are released and error E_DLT is returned when there are tasks waiting for the condition on the event flag.

8.2.4.3 ts_set_flg/tn_set_flg – Set Event Flag

C Language Interface

[Safety API]	ER ercd = ts_set_flg(ID flgid, UINT setptn);
[Normal API]	ER ercd = tn_set_flg(ID flgid, UINT setptn);

Parameter

ID	flgid	Event flag ID
UINT	setptn	Bit pattern to be set

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('flgid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the event flag specified in 'flgid' does not exist) Object is NOT existed (Event flag set by 'flgid' is NOT existed)
E_DACV	Domain access protection violation (Target event flag belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task-Independent portion
○	○	○

Description

Among bit patterns of event flags (EventFlagPattern) specified in flgid, the bit that is set to 1 in setptn will be set.
In other words, the values of the logicals of Event Flag Pattern and setptn are used as a new bit pattern of the specified event flag.When describing this as C Language, it should be expressed as below.

```
EventFlagPattern |= setptn;
```

By executing this API, the specified event flag bit patterns are changed due to the conditions being met for the release of the wait for tasks in WAITING state by the specified event flag in ts_wai_flg/tn_wai_flg. These task event flags in WAITING state are released.The task released from state of waiting for event flag is transited to running state (RUNNING) or ready state (READY).If its task on waiting and suspended state (WAITING – SUSPENDED), it is transited to suspended state (SUSPENDED).

If all the bits of setptn are cleared to 0, no operation is made to the target event flag. No error will result in either case.
Multiple tasks can wait for a single event flag if that event flag has the TA_WMUL attribute. The event flag in that case has a

queue for the waiting tasks. In this case, a single execution of this API may result in the release of multiple waiting tasks.Uses the bit patterns that have been renewed by this API and checks the wait release condition of tasks connected to the queue of the specified event flags in turn..

If the wait of a task with an event flag in WAITING state is released, and "Clear the flags where release conditions are met" is specified, the specified event flag bit pattern will change after the wait of corresponding task has been released. From here on, the changed bit patterns will be used for processing and this does not always mean that the WAITING state of the tasks connected to the queue will be released.Note that after a task has been released from its wait, the parameter in ts_wai_flg/tn_wai_flg will specify whether or not the bit pattern will be cleared. For details, refer to '8.2.4.4 ts_clr_flg/tn_clr_flg – Clear Event Flag'.

8.2.4.4 ts_clr_flg/tn_clr_flg – Clear Event Flag

C Language Interface

[Safety API]	ER ercd = ts_clr_flg(ID flgid, UINT clrptn);
[Normal API]	ER ercd = tn_clr_flg(ID flgid, UINT clrptn);

Parameter

ID	flgid	Event flag iD
UINT	clrptn	Bit pattern to be cleared

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('flgid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the event flag specified in 'flgid' does not exist) Object is NOT existed (Event flag set by 'flgid' is NOT existed)
E_DACV	Domain access protection violation (Target event flag belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

Among bit patterns of event flags (EventFlagPattern) specified in flgid, the bit that is set to 0 in clrptn will be cleared. In other words, the logical product of EventFlagPattern and clrptn is used as a new bit pattern of the specified event flag.When describing this as C Language, it should be expressed as below.

```
EventFlagPattern &= clrptn;
```

As a result of change in the specified event flag bit patterns by executing this API, no new bits will be set. This never results in wait conditions being released for a task waiting for the specified event flag; that is, dispatching never occurs.
If all the bits of clrptn are set to 1, no operation is made to the target event flag. No error will result in either case.

8.2.4.5 ts_wai_flg/tn_wai_flg – Wait Event Flag

C Language Interface

[Safety API]	ER ercd = ts_wai_flg(ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout);
[Normal API]	ER ercd = tn_wai_flg(ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout);

Parameter

ID	flgid	Event flag ID
UINT	waiptn	Wait bit pattern
UINT	wfmode	Wait mode
UINT*	p_flgptn	Pointer to the area to return the return parameter flgptn
TMO	tmout	Timeout (ms)

Return parameter

ER	ercd	Error code
UINT	flgptn	Bit pattern of wait releasing

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('flgid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the event flag specified in 'flgid' does not exist)
E_PAR	Parameter error ('waiptn=0', 'wfmode' is invalid, 'tmout' is invaid, Address set by p_flgptn' invalid)
E_OBJ	Invalid object state (multiple tasks are waiting for an event flag with TA_WSGL attribute)
E_DLT	The object being waited for was deleted (the specified event flag was deleted while waiting)
E_RLWAI	Waiting state released (ts_rel_wai received in waiting state)
E_DISWAI	Wait released due to disabling of wait
E_TMOUT	Polling failed or timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion , or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation (Target event flag belongs to other domain and the access to it is protected)
E_MACV	Memory access privilege error (There are no access privileges to 'pk_cflg' or there is no specified memory space.)
E_DOMAIN	Domain error

Valid Context

Task portion	Time Event Handler	Task-Independent portion
○	×	×

Description

Waits for the event flag bit pattern specified in flgid to be set, fulfilling the wait release condition.
If the specified event flag bit pattern already meets the wait release condition set in wfmode, the waiting task executed by this API continues executing without going to WAITING state.
The method for clearing bits for when the waiting condition for an event flag is met and for when the wait release condition is met, is specified in wfmodel.

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

TWF_ANDW	AND wait condition
TWF_ORW	OR wait condition
TWF_CLR	Clear all bitsTWF_BITCLR Clear condition bit only

The event flag waiting mode is defined in the header file as shown below.

```
#define TWF_ANDW        (0x00000001U)   /* AND wait */
#define TWF_ORW         (0x00000002U)   /* OR wait */
#define TWF_CLR         (0x00000010U)   /* All clear specify */
#define TWF_BITCLR      (0x00000020U)   /* Only condition bit clear specify */
```

If TWF_ORW is specified, the issuing task waits for any of the bits specified in waiptn to be set for the event flag bit pattern (EventFlagPattern) specified in flgid (OR wait).If TWF_ANDW is specified, the issuing task waits for all bits specified in waiptn to be set for the event flag bit pattern specified in flgid (AND wait).

The event flag bit pattern does not change, if TWF_CLR or TWF_BITCLR is not specified and the specified task changes to wait release state because the event flag wait release condition is met.The specified event flag bit pattern (all bits) are cleared by 0, if TWF_CLR is specified and the specified task changes to wait release state because the event flag wait release condition is met.Among the specified event flag bit patterns, only the bits that are set to 1 in waiptn are cleared by 0, if TWF_BITCLR is specified and the specified task changes to wait release state because the event flag wait release condition is met.When describing this as C Language, it should be expressed as below.

```
EventFlagPattern &= ~waiptn;            /* When TWF_BITCLR is specified */
```

The return parameter flgptn sets the event flag bit pattern (if TWF_CLR or TWF_BITCLR is specified, the bit pattern before the event flag is cleared) after the WAITING state of a task has been released due to this API.
In other words, the value returned by flgptn becomes the bit pattern that was specified by the wait release condition in waiptn when executing this API.flgptn are indeterminate if the event flag waitin state is released due to timeout or the like.

The timeout is set in tmout. If the during the specified time the wait release condition has not been met, error E_TMOUT is returned.When specifying tmout as TMO_FEVR (= -1), the time until timeout will be eternal. In this case, the task stays in waiting state until s_set_flg/tn_set_flg, ts_del_flg/tn_del_flg, ts_rel_wai/tn_rel_wai, or ts_dis_wai/tn_dis_wai is issued. If TMO_FEVR is specified in the Safety API, error E_DOMAIN will be returned.When specifying tmout as TMO_POL(=0), error E_TMOUT

(polling error) is returned without going into WAITING state. In the case of a timeout, the event flag bit pattern will not be cleared even if TWF_CLR or TWF_BITCLR was specified.
If smaller than or equal to ‘−2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

Setting waiptn to 0 results in Parameter error E_PAR.If waiptn = 0 were allowed, it would not be possible to get out of WAITING state regardless of the event flag values bit pattern values and that is why this usage will result in an error.
A task cannot execute ts_wai_flg for an event flag having the TA_WSGL attribute while another task is waiting for it.
Error E_OBJ will be returned for the task issuing the subsequent ts_wai_flg/tn_wai_flg, regardless of whether that task would have gone to WAITING state; i.e., regardless of whether the wait release conditions would be met.
If an event flag has the TA_WMUL attribute, multiple tasks can wait for it at the same time.
The event flag in that case has a queue for the waiting tasks.
A single ts_set_flg/tn_set_flg call for such an event flag may result in the release of multiple waiting tasks.If multiple tasks are queued for an event flag with TA_WMUL attribute, the behavior is as follows.

- Tasks are queued in either FIFO or priority order. (Release of wait state does not always start from the head of the queue, however, depending on factors such as waiptn and wfmode settings.)
- If TWF_CLR or TWF_BITCLR was specified by a task in the queue, all of the event flag bit patterns or a part of them will be cleared after that task is released from WAITING state.Tasks later in the queue than a task specifying TWF_CLR or TWF_BITCLR will decide the usage of the wait release condition after the event flag has been cleared.
- If multiple tasks having the same priority are released from waiting simultaneously as a result of ts_set_flg, the order of tasks in the ready queue (precedence) after release will continue to be the same as their original order in the event flag queue.

8.2.4.6 ts_ref_flg/tn_ref_flg – Reference Event Flag Status

C Language Interface

[Safety API]	ER ercd = ts_ref_flg(ID flgid, T_RFLG *pk_rflg);
[Normal API]	ER ercd = tn_ref_flg(ID flgid, T_RFLG *pk_rflg);

Parameter

ID	flgid	Event flag ID
T_RFLG*	pk_rflg	Pointer to the area to return the event flag status

Return parameter

ER	ercd	Error code
T_RFLG	rflg	Event flag state

State of event flag

```
typedef     struct st_rflg {
            ID          dmnno;              /* Domain number */
            ID          wtsk;               /* Waiting task ID */
            UINT        flgptn;             /* The current event flag bit pattern } T_RFLG;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('flgid' is invalid or cannot be used)
E_NOEXS	Object does not exist (the event flag specified in 'flgid' does not exist)E_PAR Parameter error (Address set by 'pk_rflg' is invalid)
E_DACV	Domain access protection violation (Target event flag belongs to other domain and the access to it is protected)
E_MACV	Memory access privilege error (There is are no access authority privileges to 'pk_cflg'. Oor there is no target specified memory areaspace.)

Valid Context

Task portion	Time event handler	Task-Independent portion
○	○	○

Description

References the status of the event flags specified in flgid and returns them as a return parameter. dmnno returns the domain number it belongs to. wtsk returns the ID of a task waiting for this event flag. If there are two or more such tasks, the ID of the task at the head of the queue is returned. If there are no waiting tasks, wtsk = 0 is returned. flgptn returns the current event flag bit pattern.

8.2.5 Synchronization and Communication Functions (Mutex)

Refer to ‘3.3.3 Mutex’ about the detail of Synchronization and Communication Functions of mutex.

8.2.5.1 ts_cre_mtx/tn_cre_mtx – Create Mutex

C Language Interface

[Safety API] ID mtxid = ts_cre_mtx(CONST T_CMTX *pk_cmtx);

[Normal API] ID mtxid = tn_cre_mtx(CONST T_CMTX *pk_cmtx);

Parameter

CONST T_CMTX* pk_cmtx Pointer to the area of information about the mutex to be created

Information about the mutex to be created

```
typedef struct st_cmtx {
        ATR     mtxatr;         /* Mutex attributes */
        PRI     ceilpri;        /* Upper limit priority for Mutex */
        UB      oname[8];       /* Object name */
} T_CMTX;
```

Return Parameter

ID	mtxid	Mutex ID
	or	Error code

Error code

E_LIMIT	Number of Mutex(s) exceeds the system limit
E_RSATR	Reserved attribute error ('mtxatr' is invalid or cannot be used)
E_PAR	Parameter error (Address set by 'pk_cmtx' is invalid, 'ceilpri' is invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_ONAME	Object name error (Specified object name has been already used in the domain)
E_MACV	Memory access violation (There is no access authority to 'pk_cmtx'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API creates Mutex and assigns Mutex ID.

'mtxatr' : Set mutex attributes for an operation for the mutex.

'mtxatr' is set as follows.

mtxatr:= (TA_TFIFO || TA_TPRI || TA_INHERIT || TA_CEILING) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	Set a queue on FIFO Tasks are queued in FIFO order
TA_TPRI	Tasks are queued in priority order
TA_INHERIT	Priority inheritance protocol
TA_CEILING	Priority ceiling protocol
TA_ONAME	Specifies object name
TA_NODISWAI	Disabling of wait by 'ts_dis_wai' is prohibited

Mutex attributes is defined in header file as follows.

```
#define TA_TFIFO        (0x00000000U)   /* manage queue by FIFO */
#define TA_TPRI         (0x00000001U)   /* manage queue by priority */
#define TA_INHERIT      (0x00000002U)   /* Priority inheritance protocol */
#define TA_CEILING      (0x00000003U)   /* Priority ceiling protocol */
#define TA_ONAME        (0x00000040U)   /* Specifies the object name */
#define TA_NODISWAI     (0x00000080U)   /* reject request to disable wait */
```

If setting 'TA_TFIFO', queue for Mutex is on the FIFO basis, 'TA_TPRI', 'TA_INHERIT' and 'TA_CEILING' on the task priority order basis.
If setting 'TA_INHERIT', priority inheritance protocol is applied, and ceiling priority protocol for 'TA_CEILING'.

'ceilpri' : Valid only if setting 'TA_CEILING'.
'ceilpri' is set with the upper limit priority for Mutex.
However the upper limit priority which exceeds the highest execution priority on the domain which the mutex belongs to cannot be set. It is possible for 'ceilpri' to set the value between ' the highest execution priority on the belonging domain ' and 'TS_LST_PRI'.

'oname' : Valid if setting 'TA_ONAME'.
'oname' is set with an object name of the Mutex.

If setting 'TA_NODISWAI', prohibiting waiting by 'ts_dis_wai/tn_dis_wai' is rejected.

8.2.5.2 ts_del_mtx/tn_del_mtx – Delete Mutex

C Language Interface

[Safety API]	ER ercd = ts_del_mtx(ID mtxid);
[Normal API]	ER ercd = tn_del_mtx(ID mtxid);

Parameter

ID	mtxid	Mutex ID

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mtxid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The mutex specified in 'mtxid' does not exist)
E_DACV	Domain access protection violation (Target mutex belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API deletes Mutex set by 'mtxid'.

The task waiting for meeting condition for the Mutex is released and error 'E_DLT' is notified.

If the Mutex with TA_INHERIT' or 'TACEILING' attributes, locked task priority is changed in some cases

8.2.5.3 ts_loc_mtx/tn_loc_mtx – Lock Mutex

C Language Interface

[Safety API]	ER ercd = ts_loc_mtx(ID mtxid, TMO tmout);
[Normal API]	ER ercd = tn_loc_mtx(ID mtxid, TMO tmout);

Parameter

ID	mtxid	Mutex ID
TMO	tmout	Timeout (ms)

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mtxid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The mutex specified in 'mtxid' does not exist)
E_PAR	Parameter error ('tmout' is invalid)
E_DLT	Waiting object was delete (Target Mutex deleted on waiting)
E_RLWAI	Release waitng state forcibly ('ts_rel_wai/tn_rel_wai' is received in waiting state)
E_DISWAI	Wait released due to disabling of wait
E_TMOUT	Polling failed or timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_ILUSE	Used illegally (Multiple locks, upper limit priority is Violated)
E_DACV	Domain access protection violation (Target mutex belongs to other domain and the access to it is protected)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API locks Mutex set by 'mtxid'.

If the mutex can be locked immediately, the task issuing this API continue executing without entering WAITING state. Mutex is on locked state.

If the Mutex already is locked, API transits the task (as issuer) to waiting state. Mutex is queued.

If the invoking task has already locked the specified mutex, error code 'E_ILUSE' (multiple lock) is returned.

The timeout is set in 'tmout'. Timeout error 'E_TMOUT' is notified if not locking before passing the timeout time.
Timeout time is infinite if setting 'TMO_FEVR(=-1) to 'tmout'. In this case the task state becomes waiting state until issuing 'ts_wup_tsk/tn_wup_tsk'. If 'TMO_FEVR' is specified in Safety API, domain error 'E_DOMAIN' is notified..
Error 'E_TMOUT' (polling error) is notified ,without transiting to waiting state, if 'TMO_POL(=0)' is set to 'tmout'.
If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

If Mutex with 'TA_INHERIT' attributes and current priority of the task locking that Mutex is lower than that of the invoking task when transiting the invoking task to waiting for lock, API sets same priority as the invoking task to its locked task. If the task waiting for lock is released from the waiting state before getting locked (such as timeout and so on), API sets the highest priority among tasks described below to the task locking that Mutex..

a. he highest priority among the current priorities of tasks waiting to lock the mutex.
b. The highest priority among all the other mutexes locked by the task currently locking this mutex
c. The base priority of the locking task.

In Mutex with 'TA_CEILING' attributes, if current priority of the invoking task is lower than upper limit priority (ceiling priority) of the Mutex when getting the lock, API sets upper limit priority to that of the invoking task. Error 'E_ILUSE' (Upper limit priority (Ceiling priority) illegal) is notified if base priority of the invoking task is higher than ceiling priority of the Mutex.
Base priority of the invoking task means a task priority before updating the priority by the Mutex automatically. Base priority is the priority set by 'ts_chg_pri/tn_chg_pri' lastly (the term including on lock by Mutex) or the task priority set at the time of creation (in case of not issuing 'ts_chg_pri/tn_chg_pri').

[Supplement]
Reason for prohibiting multiple lock for Mutex is described below.
It is possible for only the task locking the Mutex to unlock. If the task that already locked the Mutex tries to lock same Mutex and then transits to waiting state, it is not possible for the Mutex to unlock. Therefore, multiple lock is probited.

8.2.5.4 ts_unl_mtx/tn_unl_mtx – Unlock Mutex

C Language Interface

[Safety API]	ER ercd = ts_unl_mtx(ID mtxid);
[Normal API]	ER ercd = tn_unl_mtx(ID mtxid);

Parameter

ID	mtxid	Mutex ID

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mtxid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The mutex specified in 'mtxid' does not exist)
E_ILUSE	Illegal use (Mutex NOT locked by the invoking task)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion
E_DACV	Domain access protection violation (Target mutex belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API unlocks Mutex set by 'mtxid'.

API releases the task on the head of queue for lock on the Mutex from on waiting state to the state of getting the lock.

If a mutex that was not locked by the invoking task is specified, error 'E_ILUSE' is returned.

If there is unlocked Mutex having 'TA_INHERIT' or 'TA_CEILING', attributes task priority is changed as follows.

If no Mutex locked by the invoking task after unlock, priority of the invoking task is set the base priority.

If Mutex locked by the invoking task exist, priority of the invoking task is set the highest priority among the priority described below.

a. The highest priority among the current priority of the tasks in the queue of the mutex with the 'TA_INHERIT' attribute locked by the invoking task

b. Highest priority among upper limit priorities (ceiling priorities) of the Mutex(s) with 'TA_CEILING' attributes locked by the invoking task

c. Base priority of the invoking task

If a task exits (transits to dormant state (DORMANT) or non-existent state (NON-EXISTENT)) with locking the Mutex, all the Mutex(s) locked by the tasks are unlocked by kernel automatically.

8.2.5.5 ts_ref_mtx/tn_ref_mtx – Refer Mutex Status

C Language Interface

[Safety API]	ER ercd = ts_ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
[Normal API]	ER ercd = tn_ref_mtx(ID mtxid, T_RMTX *pk_rmtx);

Parameter

ID	mtxid	Mutex ID
T_RMTX*	pk_rmtx	Pointer to the area to return the mutex status

Return Parameter

ER	ercd	Error code
T_RMTX	rmtx	Mutex Status

State of Mutex

```
typedef struct st_rmtx {
        ID      dmnno;          /* Domain number */
        ID      htsk;           /* ID of task locking the mutex */
        ID      wtsk;           /* ID of tasks waiting to lock the mutex */
} T_RMTX;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mtxid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The Mutex specified in mtxid does not exist)
E_PAR	Parameter error (Address set by 'pk_rmtx' is invalid)
E_DACV	Domain access protection violation (Target mutex belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'pk_rmtx'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers to states of Mutex set by 'mtxid' and returns a return parameter.

'dmnno' : means a number of the domain.

'htsk' : means ID of the task locking the Mutex. 'htsk' is set '0' if no task if no task is locked.

'wtsk' : means ID of the task on waiting state for the Mutex. ID of the head task in queue is returned if multiple tasks on waiting state. 'wtsk' is set '0' if no task on waiting state.

8.2.6 API for Synchronization and Communication Functions (Message Buffer)

Refer to ‘3.3.4 Message ’ about the detail of Synchronization and Communication Functions of a message buffer.

8.2.6.1 ts_cre_mbf/tn_cre_mbf – Create Message Buffer

C Language Interface

[Safety API] ID mbfid = ts_cre_mbf(CONST T_CMBF *pk_cmbf);

[Normal API] ID mbfid = tn_cre_mbf(CONST T_CMBF *pk_cmbf);

Parameter

CONST T_CMBF* pk_cmbf Pointer to Message buffer creation information

Message buffer creation information

```
typedef     struct st_cmbf {
            ATR     mbfatr;         /* Message buffer attributes
            INT     mbfno;          /* Message buffer management number */
            INT     bufsz;          /* Message buffer size (in bytes) */
            INT     maxmsz;         /* Maximum message size (in bytes) */
            UB      oname[8];       /* Object name */
} T_CMBF;
```

Return Parameter

ID	mbfid	Message buffer ID
	or	Error code

Error code

E_LIMIT	Number of message buffers be beyond the upper limit on the system
E_RSATR	Reserved attribute error ('mbfatr' is invalid or cannot be used)
E_PAR	Parameter error (Address set by 'pk_cmbf' is invalid, 'mbfno','bufsz', 'maxmsz' are negative or invalid
E_NOMEM	Insufficient memory error ('bufsz' is larger than memory area which message buffer can use)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion
E_ONAME	Object name error (Specified object name has been already used in the domain)
E_MACV	Memory access violation (There is no access authority to 'pk_cmbf ', no target memory area. or memory area specified by 'mbfno' is not the data area of the self domain)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API creates a message buffer and assigns the ID number of message buffer.

'mbfatr' : Set message buffer attributes for setting the operation of the message buffer.

'mbfatr' is set as follows.

mbfatr := (TA_TFIFO || TA_TPRI) | [TA_ONAME] | [TA_NODISWAI]

TA_TFIFO	Task on waiting for sending queued on FIFO basis
TA_TPRI	Task on waiting for sending queued on priority order basis
TA_ONAME	Specifies the object name
TA_NODISWAI	Reject disabling wait by 'ts_dis_wai/tn_dis_wai'

Message buffer attributes are defined in header file as follows.

```
#define TA_TFIFO        (0x00000000U)   /* Manage task queue by FIFO */
#define TA_TPRI         (0x00000001U)   /* Manage task queue by priority */
#define TA_ONAME        (0x00000040U)   /* Specifies the object name */
#define TA_NODISWAI     (0x00000080U)   /* Reject request to disable wait */
```

By 'TA_TFIFO' and 'TA_TPRI', it is possible for the task to set the order in queue if message buffer full. 'TA_TFIFO' is set for a queue on FIFO basis, 'TA_TPRI' for a queue on priority order basis.
The order of queue of task for receiving message is FIFO basis only.

'oname' : Valid if setting 'TA_ONAME'.
'oname' is specified with the object name of the message buffer.

Disabling wait by 'ts_dis_wai/tn_dis_wai' is rejected if setting 'TA_NODISWAI'.

The number of message buffer management information specified by configuration is specified to 'mbfno'. The information of memory resource used by message buffer is spscified to message buffer management information.
However if bufsz equal '0', 'mbfno' does not need to be specified. Specified value is ignored.
If the number which has been used by the other message buffer is specified, error 'E_PAR' is returned.
And if memory area which cannot be used by the message buffer created is specified to specified message buffer management information (the memory area is not the area of the self domain), error 'E_MACV' is returned.

Size (byte number) of buffer area used by the target message buffer is specified to 'bufsz'.
Not only the message but also the information to manage the individual message are stored to used buffer area. For this reason buffer size specified by 'bufsz' and the total of message size in the queue are not matched. The latter value is smaller.
The value which can be specified to 'bufsz' is more than '0'. Negative value cannot be set. Maximum value should be less than the size of memory area of message buffer management information specified by 'mbfno'
If the specified value exceeds the size of memory area of message buffer management information, error 'E_NOMEM' is returned.
It is possible to create message buffer whose 'bufsz' equal '0'. In this case in this message buffer, communication of transmission and reception side is completely synchronized. Therefore if the system call of ether 'ts_snd_mbf/tn_snd_mbf' or

‘ts_rcv_mbf/tn_rcv_mbf’ is executed ahead, the task executing the system call becomes wating state. At the timing when the other system call is executed, message is passed (copied) and then both tasks resume the execution.

Maximum length (byte number) of message to send and receive is specified to ‘maxmsz’. The valule of more than ‘1’ is specified to ‘maxmsz’. Maximum value is specified by configuration.
At the transmission side the size of transmission message should be less than the value specified by ‘maxmsz’. If the size of transmission message exceeds ‘maxmsz’, error occurs.
At the reception side the reception buffer area whose size is more than the byte number specified by ‘maxmsz’ should be prepared.

‘maxmsz’ may specify the value which is larger than ‘bufsz’. For example, if the message whose size is larger than ‘bufsz’ tries to be sent to the message buffer whose ‘maxmsz’ is larger than ‘bufsz’, sent message cannot be stored in the buffer area. Then if there is some task on waiting state for receiving, the message is copied from transmission buffer to reception buffer directly. On the other hand in the target message buffer, if there is no task on waiting state for receiving, after the task of transmission side becomes on waiting state for sending for a moment, when reception of message to the target message buffer is executed, the message is copied from transmission buffer to reception buffer directly. However if message has been already stored in the buffer area, the stored message should be all processed first. (the order of the message queue is only FIFO.)

8.2.6.2 ts_del_mbf/tn_del_mbf – Delete Message Buffer

C Language Interface

[Safety API]	ER ercd = ts_del_mbf(ID mbfid);
[Normal API]	ER ercd = tn_del_mbf(ID mbfid);

Parameter

ID	mbfid	Message buffer ID

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mbfid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The message buffer specified in mbfid does not exist)
E_DACV	Domain access protection violation (Target message buffer belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API deletes the message buffer set by 'mbfid'.

Message buffer is deleted, error not notified, even if leaving any messages in the message buffer.

Task on waiting state for the message buffer is released and error 'E_DLT' is notified.

8.2.6.3 ts_snd_mbf/tn_snd_mbf – Send Message to Message Buffer

C Language Interface

[Safety API]	ER ercd = ts_snd_mbf(ID mbfid, CONST void *msg, INT msgsz, TMO tmout);
[Normal API]	ER ercd = tn_snd_mbf(ID mbfid, CONST void *msg, INT msgsz, TMO tmout);

Parameter

ID	mbfid	Message buffer ID
CONST void*	msg	Start address of send message
INT	msgsz	Size of sending message (by bytes)
TMO	tmout	Timeout (ms)

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mbfid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The message buffer specified in 'mbfid' does not exist)
E_PAR	Parameter error (Address set by 'msg' is invalid, 'msgsz=<0', 'maxmsz<msgsz', 'tmout' is invalid)
E_DLT	Waiting object was deleted (Message buffer was deleted while waiting)
E_RLWAI	Waiting state was forcibly released ('ts_rel_wai' is received on waiting)
E_DISWAI	Wait released due to disabling of wait
E_TMOUT	Polling failed or timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation (Target message buffer belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'msg', or there is no target memory area.)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API sends a message to the message buffer set by 'mbfid'.

'msg' : Set a pointer to the memory area storing a sending message.

'msgsz' : Set a message size (by bytes). It is possible to set the value between '1' and 'maxmsz'.
Message in size 'msgsz' byte(s), address starting from 'msg' is copied to the message buffer set by 'mbfid', and then set the queue for the message.
If not possible to copy a message to buffer area because of insufficient buffer space, the task issuing this API is transited to waiting state for sending the message and set the queue for getting a buffer space for sending (Queue for sending). Order of the queue for sending is indicated by 'ts_cre_mbf/tn_cre_mbf either the one on FIFO basis or task priority order basis'.

'tmout' : Set timeout time. Timeout error 'E_TMOUT' is notified if not sending a message before passing the timeout time.
Timeout time is infinite if setting 'TMO_FEVR(=-1) to 'tmout'. API waits until sending a message. If 'TMO_FEVR' is specified in Safety API, domain error 'E_DOMAIN' is notified.
Error 'E_TMOUT' (polling error) is notified without transiting to waiting state, if 'TMO_POL(=0)' to 'tmout'.
If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

8.2.6.4 ts_rcv_mbf/tn_rcv_mbf – Receive Message from a Message Buffer

C Language Interface

[Safety API]	INT	msgsz = ts_rcv_mbf(ID mbfid, void *msg, TMO tmout);
[Normal API]	INT	msgsz = tn_rcv_mbf(ID mbfid, void *msg, TMO tmout);

Parameter

ID	mbfid	Message buffer ID
void*	msg	Address for receiving message
TMO	tmout	Timeout (ms)

Return parameter

INT	msgsz	Size of receiving message (by bytes)
	or	Error code

Error code

E_ID	Invalid ID number ('mbfid' invalid or cannot be used)
E_NOEXS	Object does not exist (The message buffer specified in mbfid does not exist)
E_PAR	Parameter error (Address set by 'msg' Invalid, 'tmout' is invalid)
E_DLT	Waiting object was deleted (Message buffer was deleted while waiting)
E_RLWAI	Waiting state released ('ts_rel_wai/tn_rel_wai' is received in waiting state)
E_DISWAI	Wait released due to disabling of wait
E_TMOUT	Polling failed or timeoutPolling failed or timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation (Target message buffer belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'msg'. Or there is no target memory area.)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API receives a message from the message buffer set by 'mbfid' and sets its message to the memory area set by 'msg'.

The head message in the message queue for the message buffer is copied to the memory area set by 'msg', and then deleted from its message queue.

'msg' : Set a pointer to the memory area for storing a receiving message.

Memory space set by 'msg' shall be reserved in the memory size larger than the size set by 'maxmsz' on 'ts_cre_mbf/tn_cre_mbf'.

If the message queue for the message buffer has no message, the task issuing this API is transited to waiting state for receiving a message and set the queue for arriving a message (Queue for receiving). Order of the queue for receiving is indicated the one on FIFO basis only.

'tmout' : Set timeout time.

'tmout' is set timeout time. Timeout error 'E_TMOUT' is notified if not receiving a message before passing the timeout time. Timeout time is infinite if setting 'TMO_FEVR(=-1) to 'tmout'. API waits until receiving a message. If 'TMO_FEVR' is specified in Safety API, domain error 'E_DOMAIN' is notified.

Error 'E_TMOUT' (polling error) is notified without transiting to waiting state, if 'TMO_POL(=0)' to 'tmout'.

If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

8.2.6.5 ts_ref_mbf/tn_ref_mbf – Reference Message Buffer Status

C Language Interface

[Safety API] ER ercd = ts_ref_mbf(ID mbfid, T_RMBF *pk_rmbf);

[Normal API] ER ercd = tn_ref_mbf(ID mbfid, T_RMBF *pk_rmbf);

Parameter

ID	mbfid	Message buffer ID
T_RMBF*	pk_rmbf	Pointer to the area returning a state of message buffer

Return Parameter

ER	ercd	Error code
T_RMBF	rmbf	Message buffer state

Message buffer state

```
typedef struct st_rmbf {
        ID      dmnno;          /* Domain number */
        ID      wtsk;           /* ID of the task on waiting for receiving */
        ID      stsk;           /* ID of the task on waiting for sending */
        INT     msgsz;          /* Size of the message sending at the next time (by bytes) */
        INT     frbufsz;        /* Size of free buffer space (by bytes) */
        INT     maxmsz;         /* Maximum length of message */
} T_RMBF;
```

Error code

E_OK	Normal completion
E_ID	Invalid ID number ('mbfid' is invalid or cannot be used)
E_NOEXS	Object does not exist (The message buffer specified in mbfid does not exist)
E_PAR	Parameter error (Address set by 'pk_rmbf' is invalid)
E_DACV	Domain access protection violation (Target message buffer belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'pk_rmbf'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API refers to state of message buffer set by 'mbfid' and returns it as a return parameter.

'dmnno' : means a number of domain belonged to.

'wtsk' : means ID of the task on the head of the queue for receiving on the message buffer. 'wtsk' is set '0' if no task.

'stsk' : means ID of the task on the head of the queue for sending on the message buffer. 'stsk' is set '0' if no task.

'msgsz' : means a size of the head message in queue for receiving (message received at the next time). 'msgsz' is set '0' if no message.

'frbufsz' : means free size of buffer area for the message queue. Approximately possible size of sending message is indicated by this value. Message in size 'frbufsz' is not sent in some cases because of its size including information for message management.

'maxmsz' : means the maximum length of message set by 'ts_cre_mbf/tn_cre_mbf'.

8.2.7 API for Time Management Functions

Refer to ‘3.4 Time Management Function’ about the detail of time management functions.

8.2.7.1 ts_set_tim/tn_set_tim – Set Time

C Language Interface

[Safety API]	ER ercd = ts_set_tim(CONST SYSTIM *pk_tim);
[Normal API]	ER ercd = tn_set_tim(CONST SYSTIM *pk_tim);

Parameter

CONST SYSTIM*	pk_tim	Pointer to the packet 'tim' indicating current time (by milliseconds)

Contents of 'tim'

W	hi	Upper 32 bits of current time for setting system time
UW	lo	Lower 32 bits of current time for setting system time

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error (Address set by 'pk_tim' is invalid, setting time is invalid
E_MACV	Memory access violation (There is no access authority to 'pk_tim'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API is set the value set by 'tim' to system time as current time.

System time is measured from 1985/1/1 00:00:00 000 (GMT) by milliseconds in 64 bit signed integer.

'tim' : Possilble to set the value between '0' and '9,223,372,036,854,775,807 (=0x7FFFFFFFFFFFFFFF)'. Error 'E_PAR' is notified if setting negative value.

Relative time set by 'RELTIM' or 'TMO' is not changed if setting system time on operating the system.

For example, if setting timeout after passing 60 seconds and then setting forward time by 60 seconds to system time, API executes timeout after passing 60 seconds. System time for timeout is changed.

[Supplement]

Time set by this API is not depended on cycle time of system timer. After its setting, time by 'ts_get_tim/tn_get_tim' is depended on time resolution of system timer. For example, if setting 5 milliseconds to time on cycle timer 10 milliseconds, time by 'ts_get_tim/tn_get_tim' is changed such as 5 milliseconds -> 15 milliseconds -> 25 milliseconds.

8.2.7.2 ts_get_tim/tn_get_tim – Get Time

C Language Interface

[Safety API]	ER ercd = ts_get_tim(SYSTIM *pk_tim);
[Normal API]	ER ercd = tn_get_tim(SYSTIM *pk_tim);

Parameter

SYSTIM*	pk_tim	Pointer to the area returning current time 'tim'

Return parameter

ER	ercd	Error code
SYSTIM	tim	Current time

Contents of 'tim'

W	hi	Upper 32 bits of current time on system time
UW	lo	Lower 32 bits of current time on system time

Error code

E_OK	Normal completion
E_PAR	Parameter error (Address set by 'pk_tim' is invalid
E_MACV	Memory access violation (There is no access authority to 'pk_tim'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers to current time of system time and returns a return parameter 'tim'.

System time is counted from 1985/1/1 00:00:00 000 (GMT) by milliseconds in 64 bit signed integer.

[Supplement]

Resolution of current time referred by this API is depended on time resolution for cycle time of system timer, defined on implementation-defined.

It is NOT possible to refer time shorter than cycle time of system timer.

8.2.7.3 ts_get_otm/tn_get_otm – Get Operating Time

C Language Interface

[Safety API]	ER ercd = ts_get_otm(SYSTIM *pk_tim);
[Normal API]	ER ercd = tn_get_otm(SYSTIM *pk_tim);

Parameter

SYSTIM*	pk_tim	Packet of Operating Time	Pointer to the area returning operating time 'tim'

Return parameter

ER	ercd	Error code
SYSTIM	tim	Operating time

Contents of 'tim'

W	hi	Upper 32 bits of system operating time
UW	lo	Lower 32 bits of system operating time

Error code

E_OK	Normal completion
E_PAR	Parameter error (Address set by 'pk_tim' is invalid
E_MACV	Memory access violation (There is no access authority to 'pk_tim'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API refers to system operating time and returns the return parameter 'tim'.

System operating time means the time measured after starting the system, not depended on time set by 'ts_set_tim/tn_set_tim', different from system time.

System operating time has same resolution as that of system time.

8.2.7.4 ts_cre_cyc/tn_cre_cyc – Create Cyclic Handler

C Language Interface

[Safety API] ID cycid = ts_cre_cyc(CONST T_CCYC *pk_ccyc);

[Normal API] ID cycid = tn_cre_cyc(CONST T_CCYC *pk_ccyc);

Parameter

CONST T_CCYC* pk_ccyc Pointer to the definition information for cyclic handler

Definition information for cyclic handler

```
typedef struct st_ccyc {
        ATR      cycatr;       /* Attributes of cyclic handler */
        INT      stacd;        /* Starting code of cyclic handler */
        FP       cychdr;       /* Start-up address of cyclic handler
        UINT     ustkno;       /* Resource number of user stack */
        INT      ustksz;       /* User stack size (byte number) */
        UINT     sstkno;       /* Resource number of system stack */
        INT      sstksz;       /* System stack size (byte number) */
        RELTIM   cyctim;       /* Cycle starting interval (by milliseconds) */
        RELTIM   cycphs;       /* Cycle starting phase (by milliseconds) */
        RELTIM   maxrtim;      /* Maximum consecutive execution time */
        UB       oname[8];     /* Object name */
} T_CCYC;
```

Return parameter

ID	cycid	Cyclic handler ID
	or	Error code

Error code

E_NOMEM	Insufficient memory error (Stack of specified size cannot be allocated from stack area)
E_LIMIT	Number of cyclic handlers be beyond the upper limit on the system
E_RSATR	Attributes reserved error ('cycatr' is invalid or NOT used)
E_PAR	Parameter error (Addresses set by 'pk_ccyc', 'cychdr' are invalid or 'maxrtim' is larger than upper limit of continuous run time or 'cyctim')
E_CTX	Context error (Issued in the state other than a task portion and a quasi-task portion)
E_ONAME	Object name error (Specified object name has been already used in the domain)
E_MACV	Memory access violation (There is no access authority to 'pk_ccyc' or 'cychdr'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API creates a cyclic handler and assigns a cyclic handler ID.

'cycatr' : Set an attributes of cyclic handler for indicating the operation of the cyclic handler.
'cycatr' is set as follows.

cycatr := [TA_STA] | [TA_PHS] | [TA_ONAME]

TA_STA	Start up after creating a cyclic handler
TA_PHS	Store a phase at starting
TA_ONAME	Set an object name

```
#define TA_STA      (0x00000002U)   /* Start a cyclic handler */
#define TA_PHS      (0x00000004U)   /* Store a cyclic handler starting phase */
#define TA_ONAME    (0x00000040U)   /* Set an object name */
```

If setting 'TA_STA', cyclic handler is transited running state at creating and executed at the setting intervals.
If not setting 'TA_STA', cyclic handler is transited stopped state and cycle time is measured (cyclic handler not started).

Cyclic handler is coded as function in C language described below.

```
void        cychdr( INT stacd);
```

Cyclic handler start-up address 'cychdr' is set the pointer to the function described above.
Cyclic handler starting code 'stacd' is passed to an argument as cyclic handler starting parameter.

Cyclic handler has user stack and system stack. However cyclic handler of System domain has only system stack and does not have user stack. So to specify user stack is ignored in creating cyclic handler of System domain.
The memory of user stack is allocated from the stack area of domain which cyclic handler belongs to. Stack resource number specified by configuration is specified to 'ustkno'. User stack size is specified to 'ustksz'. However since memory area is allocated dynamically in Normal domain, the value of 'utskno' is ignored in Normal API. In Safety API 'ustksz' is less than the memory size of the stack resource specified by 'ustkno'.
The memory of system stack is allocated from the stack area of System domain. Stack resource number specified by configuration is specified to 'sstkno'. System stack size is specified to 'sstksz'.' sstksz' is less than the memory size of the stack resource specified by 'sstkno'.
If stack cannot be allocated for the following reason, error 'E_NOMEM' is notified.

- Specified stack size is larger than the size of stack resource.
- Specified stack resource has been already used by task or time event handler.
- User stack cannot be allocated in Normal API (Rest size of user stack area is lacked)

'cyctim' : Set an interval for the cycle (by milliseconds), 'cycphs' a phase for the cycle (by milliseconds).
It is possible for 'cyctim' to set the value between '1' and 'MAX_CYCTIM', 'cycphs' the value between '0' and 'TS_MAX_CYCPHS'. 'TS_MAX_CYCTIM' and 'TS_MAX_CYCPHS' is implementation-defined.
Starting time of cyclic handler is indicated by 'cyctim' and 'cycphs'. Cyclic handler starts at the time passing 'cycphs' after creating, and then operates repeated at the 'cyctim' intervals.
If setting '0' to 'cycphs', cyclic handler starts immediately after creating. It is NOT possible to set '0' to 'cyctim'. The n-th starting time of a cyclic handler after creating is corresponded to 'cycphs + cyctim * (n-1)' and over.
If not setting 'TA_STA', cycle time is measured (cyclic handler not started).

If setting 'TA_PHS' and then issued 'ts_sta_cyc/tn_sta_cyc', cyclic handler starting phase is kept. If the cyclic handler transited to running by 'ts_sta_cyc/tn_sta_cyc', the cycle time is not reset, continued to start at the initial intervals (measuring from at creating).
If not setting 'TA_PHS', cycle time is reset by 'ts_sta_cyc/tn_sta_cyc' and then cyclic handler is started in the interval 'cyctim' after calling 'ts_sta_cyc/tn_sta_cyc'. 'cycphs' is not applied for the reset. The n-th starting time of cyclic handler after calling by 'ts_sta_cyc/tn_sta_cyc' is corresponds to 'cyctim * n' and over.

'maxrtim' : Set the maximum consecutive execution time for cyclic handler by milliseconds. Maximum value is the upper limit of consecutive execution time on the self domain. 'maxrtim' shall be lower than that of 'cyctim'.
Error 'E_PAR' is notified if setting over the upper limit of consecutive execution time on the self domain to 'maxrtim' or if setting larger value than that of 'cyctim'.
API notifies an abnormality exception and suspends an execution of cyclic handler if cyclic handler operates consecutively over 'maxrtim',

'dsname' : Valid if setting 'TA_ONAME'.

'oname' : Set an object name of the cyclic handler.

Cyclic handler is executed with the highest execution priority on the self domain, prior to tasks with same priority. If it is starting time for the cyclic handler on executing a task with same priority, its task is suspended the execution and started the cyclic handler. Cyclic handler is not suspended the execution by other time event handler and task on the self domain.

[Supplement]

Cyclic handler is executed with the highest priority on the self domain and not suspended the execution. Task and time event handler on other domain are prior to execute if having higher priority. Cyclic handler on normal domain has lower priority than task and time event handler on Safety domain because of Normal domain not prior to Safety domain. Execution of the software on Ssafety domain is protected from that of Normal domain.

8.2.7.5 ts_del_cyc/tn_del_cyc – Delete Cyclic Handler

C Language Interface

[Safety API]	ER ercd = ts_del_cyc(ID cycid);
[Normal API]	ER ercd = tn_del_cyc(ID cycid);

Parameter

ID	cycid	Cyclic handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('cycid' is invalid or not used)
E_NOEXS	Object NOT existed (Cyclic handler set by 'cycid' is NOT existed)
E_DACV	Domain access protection violation (Target cyclic handler belongs to other domain and the access to it is protected)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API deletes the cyclic handler set by 'cycid'.

It is possible to delete the cyclic handler even if on running state.

8.2.7.6 ts_sta_cyc/tn_sta_cyc – Start Cyclic Handler

C Language Interface

[Safety API]	ER ercd = ts_sta_cyc(ID cycid);
[Normal API]	ER ercd = tn_sta_cyc(ID cycid);

Parameter

ID	cycid	Cyclic handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('cycid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Cyclic handler with 'cycid' is NOT existed)
E_DACV	Domain access protection violation (Target cyclic handler belongs to other domain and the access to it is protected)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API transits the cyclic handler set by 'cycid' to running state.

If setting 'TA_PHS' attributes for cyclic handler at creating, the cycle time is not reset, continued to start at the initial intervals (measuring from at creating). If cyclic handler set by 'cycid' already executed, the operation of the cyclic handler is kept.

If not setting 'TA_PHS' attributes for cyclic handler, the cycle time is reset and then transited its cyclic handler to running state. If the cyclic handler set by 'cycid' already executed, the cycle time is reset and then the operation of the cyclic handler is kept. The cyclic handler is started at the time passing 'cyctim' after issuing this API.

8.2.7.7 ts_stp_cyc/tn_stp_cyc – Stop Cyclic Handler

C Language Interface

[Safety API] ER ercd = ts_stp_cyc(ID cycid);

[Normal API] ER ercd = tn_stp_cyc(ID cycid);

Parameter

ID	cycid	Cyclic handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('cycid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Cyclic handler with 'cycid' is NOT existed)
E_DACV	Domain access protection violation (Target cyclic handler belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API stops the cyclic handler set by 'cycid'.

API has no operation if the cyclic handler already stopped.

8.2.7.8 ts_ref_cyc/tn_ref_cyc – Reference Cyclic Handler Status

C Language Interface

[Safety API]	ER ercd = ts_ref_cyc(ID cycid, T_RCYC *pk_rcyc);
[Normal API]	ER ercd = tn_ref_cyc(ID cycid, T_RCYC *pk_rcyc);

Parameter

ID	cycid	Cyclic handler ID
T_RCYC*	pk_rcyc	Pointer to the area for returning a state of cyclic handler

Return parameter

ER	ercd	Error code
T_RCYC	rcyc	Cyclic handler status

State of cyclic handler

```
typedef struct st_rcyc {
        ID      dmnno;    /* Domain number */
        PRI     cycpri;   /* Cyclic handler priority */
        RELTIM  lfttim;   /* Left time to start a cyclic handler at the next time (by milliseconds) */
        UINT    cycstat;  /* Cyclic handler status */
} T_RCYC;
```

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('cycid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Cyclic handler set by 'cycid' is NOT existed)
E_PAR	Parameter error (Address set by 'pk_rcyc' is invalid)
E_DACV	Domain access protection violation (Target cyclic handler belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'pk_rcyc'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API refers to a state of the cyclic handler set by 'cycid'.

'dmnno' : means a number of the self domain.

'cycpri' : means a cyclic handler priority. It is set the highest priority on the self domain at creating the cyclic handler.

'lfttim' : means left time to start a cyclic handler at the next time (by millisecond). It is not depended on whether the cyclic handler being on executing or not.

'cycstat' : means cyclic handler status.
Cyclic handler status is shown as follows.

cycstat:= (TCYC_STP | TCYC_STA | TCYC_EXE)

```
#define TCYC_STP    (0x00000000U)     /* Cyclic handler stopped */
#define TCYC_STA    (0x00000001U)     /* Cyclic handler started */
#define TCYC_EXE    (0x00000002U)     /* Cyclic handler executed */
```

8.2.7.9 ts_cre_alm/tn_cre_alm – Create Alarm Handler

C Language Interface

[Safety API] ID almid = ts_cre_alm(CONST T_CALM *pk_calm);

[Normal API] ID almid = tn_cre_alm(CONST T_CALM *pk_calm);

Parameter

CONST T_CALM* pk_calm Pointer to information for definition of an alarm handler

Information for definition of an alarm handler

```
typedef struct st_calm {
        ATR     almatr;         /* Attributes of alarm handler */
        INT     stacd;          /* Alarm handler starting code */
        FP      almhdr;         /* Address of alarm handler
        UINT    ustkno;         /* Resource number of user stack */
        INT     ustksz;         /* User stack size (byte number) */
        UINT    sstkno;         /* Resource number of system stack */
        INT     sstksz;         /* System stack size (byte number) */
        RELTIM  maxrtim;        /* Maximum consecutive execution time */
        UB      oname[8];       /* Object name */
} T_CALM;
```

Return parameter

ID	almid	Alarm handler ID
	or	Error code

Error code

E_NOMEM	Insufficient memory error (Stack of specified size cannot be allocated from stack area)
E_LIMIT	Number of alarm handlers be beyond the upper limit on the system
E_RSATR	Attributes reserved error ('almatr' is invalid or NOT used)
E_PAR	Parameter error (Addresses set by 'pk_calm', 'almhdr' are invalid or 'maxrtim' is larger than upper limit of continuous run time of the self domain)
E_ONAME	Object name error (Specified object name has been already used in the domain)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_MACV	Memory access violation (There is no access authority to 'pk_calm' or 'almhdr'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API creates an alarm handler and assigns an alarm handler ID.

'almatr' : Set attributes for the alarm handler for indicating the operation of the alarm handler.
'almatr' is set as follows.

almatr := [TA_ONAME]

TA_ONAME	Set an object name

```
#define TA_ONAME    (0x00000040U)        /* Set an object name */
```

Alarm handler is coded by a function in C language as follows.

```
void        almhdr( INT stacd);
```

Alarm handler address 'almhdr' is set the pointer to function described above.
Alarm handler starting code 'stacd' is passed to an argument as alarm handler starting parameter.

Alarm handler has user stack and system stack. However alarm handler of System domain has only system stack and does not have user stack. So to specify user stack is ignored in creating alarm handler of System domain.
The memory of user stack is allocated from the stack area of domain which alarm handler belongs to. Stack resource number specified by configuration is specified to 'ustkno'. User stack size is specified to 'ustksz'. However since memory area is allocated dynamically in Normal domain, the value of 'utskno' is ignored in Normal API. In Safety API 'ustksz' is less than the memory size of the stack resource specified by 'ustkno'.
The memory of system stack is allocated from the stack area of System domain. Stack resource number specified by configuration is specified to 'sstkno'. System stack size is specified to 'sstksz'.' sstksz' is less than the memory size of the stack resource specified by 'sstkno'.
If stack cannot be allocated for the following reason, error 'E_NOMEM' is notified.

- Specified stack size is larger than the size of stack resource.
- Specified stack resource has been already used by task or time event handler.
- User stack cannot be allocated in Normal API (Rest size of user stack area is lacked)

'maxrtim' : Set the maximum consecutive execution time for alarm handler by milliseconds. Maximum value is the upper limit of consecutive execution time on the self domain.
Error 'E_PAR' is notified if setting over the upper limit of consecutive execution time on the self domain to 'maxrtim'.
API notifies an abnormality exception and suspends an execution of alarm handler if alarm handler operates consecutively over 'maxrtim',

'oname' : Valid if setting 'TA_ONAME'.

'oname' is set an object name of the alarm handler.

Alarm handler is executed with the highest execution priority on the self domain, prior to tasks with same priority. If it is starting time for the alarm handler on executing a task with same priority, its task is suspended the execution and started the alarm handler. Alarm handler is not suspended the execution by other time event handler and task on the self domain.

[Supplement]

Alarm handler is executed with the highest priority on the self domain and not suspended the execution. Task and time event handler on other domain are prior to execute if having higher priority. Alarm handler on normal domain has lower priority than task and time event handler on Safety domain because of Normal domain not prior to Safety domain. Execution of the software on Safety domain is protected from that of Normal domain.

8.2.7.10 ts_del_alm/tn_del_alm – Delete Alarm Handler

C Language Interface

[Safety API]	ER ercd = ts_del_alm(ID almid);
[Normal API]	ER ercd = tn_del_alm(ID almid);

Parameter

ID	almid	Alarm handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('almid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Alarm handler set by 'almid' is NOT existed)
E_DACV	Domain access protection violation (Target alarm handler belongs to other domain and the access to it is protected)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API deletes the alarm handler set by 'almid'.

It is possible to delete the alarm handler even if on running state.

8.2.7.11 ts_sta_alm/tn_sta_alm – Start Alarm Handler

C Language Interface

[Safety API]	ER ercd = ts_sta_alm(ID almid, RELTIM almtim);
[Normal API]	ER ercd = tn_sta_alm(ID almid, RELTIM almtim);

Parameter

ID	almid	Alarm handler ID
RELTIM	almtim	Starting time of alarm handler (by milliseconds)

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('almid' is invalid or NOT used
E_PAR	Parameter error('almtim' is invalid)
E_NOEXS	Object NOT existed (Alarm handler with 'almid' is NOT existed)
E_DACV	Domain access protection violation (Target alarm handler belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API sets starting time for the alarm handler set by 'almid' and transits to running state.

'**almtim**' :Set starting time of the alarm handler.
It is possible for 'almtim' to set the value between '0' and 'TS_MAX_ALMTIM'. TS_MAX_ALMTIM is implementation-defined.
Alarm handler is started after passing the time set by 'almtim' (measuring from the time calling this API.
If setting '0' to 'almtim', alarm handler is started immediately after this API executing.

If this API already issued and then executed again for the alarm handler on ready state, API releases the starting time already set and sets the alarm handler again.

8.2.7.12 ts_stp_alm/tn_stp_alm – Stop Alarm Handler

C Language Interface

[Safety API] ER ercd = ts_stp_alm(ID almid);

[Normal API] ER ercd = tn_stp_alm(ID almid);

Parameter

ID	almid	Alarm handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('almid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Alarm handler set by 'almid' is NOT existed)
E_DACV	Domain access protection violation (Target alarm handler belongs to other domain and the access to it is protected)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API releases starting time of the alarm handler set by 'almid' and transits to stopped state.

API has no operation if already stopped.

8.2.7.13 ts_ref_alm/tn_ref_alm – Reference Alarm Handler Status

C Language Interface

[Safety API]	ER ercd = ts_ref_alm(ID almid, T_RALM *pk_ralm);
[Normal API]	ER ercd = tn_ref_alm(ID almid, T_RALM *pk_ralm);

Parameter

ID	almid	Alarm handler ID
T_RALM*	pk_ralm	Pointer to the area returning a state of alarm handler

Return parameter

ER	ercd	Error code
T_RALM	ralm	Alarm handler state

State of alarm handler

```
typedef struct st_ralm {
        ID        dmnno;     /* Domain number */
        PRI       almpri;    /*Alarm handler priority */
        RELTIM    lfttim;    /* Left time to starting an alarm handler (by milliseconds) */
        UINT      almstat;   /* Alarm handler state */
} T_RALM;
```

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('almid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Alarm handler set by 'almid' is NOT existed)
E_PAR	Parameter error (Address set by 'pk_ralm' is invalid)
E_DACV	Domain access protection violation (Target alarm handler belongs to other domain and the access to it is protected)
E_MACV	Memory access violation (There is no access authority to 'pk_ralm'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API refers states of the alarm handler set by 'almid'.

'dmnno' : means a number of the self domain.

‘almpr’ means a priority of alarm handler.

'lfttim' : means left time to start an alarm handler at the next time (by millisecond). It is undefined if on stopped state.

'almstat' : means alarm handler state.

Alarm handler state is indicated as follows.

almstat:= (TALM_STP | TALM_STA | TALM_EXE)

```
#define TALM_STP     (0x00000000U)     /* Alarm handler stopped */
#define TALM_STA     (0x00000001U)     /* Alarm handler started */
#define TALM_EXE     (0x00000002U)     /* Alarm handler executed */
```

8.2.8 API for System Configuration Information Management

Refer to ‘3.5 System Configuration Information Management Functions’ about the detail of system configuration information management.

8.2.8.1 ts_get_cfn/tn_get_cfn – Get Numbers

C Language Interface

[Safety API]	INT ct = ts_get_cfn(CONST UB *name, INT *val, INT max);
[Normal API]	INT ct = tn_get_cfn(CONST UB *name, INT *val, INT max);

Parameter

CONST UB*	name	System configuration information name
INT*	val	Array for storing numeric string of system configuration information
INT	max	Number of elements for the array set by 'val'

Return parameter

INT	ct	Number of elements for the numeric string defined on
	or	Error code

Error Code

E_NOEXS	System configuration information is NOT defined
E_OBJ	System configuration information is NOT numeric string or out of range
E_PAR	Parameter error (Addresses set by 'name',' val' are invalid, or ' max' is Negative value)
E_MACV	Memory access violation (There is no access authority to 'val'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API gets system configuration information by numeric string.
API gets numeric string of system configuration information set by 'name' in maximum number of values 'max' or less and stores to 'val'.

API returns a number of elements for defined numeric string as return value.
If the return value is bigger than 'max', all of the information is not stored.
In this case, API gets the numeric string in 'max', not get in 'max'+1 and over. If numeric string in 'max'+1 and over having illegal (such as on its format or on its value), error is not notified.

It is possible to get a number of elements for numeric string without storing to 'val' by setting '0' to 'max'.

Error 'E_NOEXS' is notified if system configuration information set by 'name' not defined.

Error 'E_OBJ' is notified if system configuration information set by 'name' not be by numeric string or not stored in INT type (Overflowed).

'val' is undefined if error notified.

8.2.8.2 ts_get_cfs/tn_get_cfs – Get Character Strings

C Language Interface

[Safety API]	INT rlen = ts_get_cfs(CONST UB *name, UB *buf, INT max);
[Normal API]	INT rlen = tn_get_cfs(CONST UB *name, UB *buf, INT max);

Parameter

CONST UB*	name	Name
UB*	buf	Array storing strings
INT	max	Maximum length of 'buf' (by bytes)

Return parameter

INT	rlen	Length of defined strings (by bytes)
	or	Error code

Error code

E_NOEXS	System configuration information is NOT defined
E_PAR	Parameter error (Addresses set by 'name','buf' are invalid, or ' max' is Negative value)
E_MACV	Memory access violation (There is no access authority to 'name'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API gets system configuration information by character string.

API gets character string of system configuration information set by 'name' in maximum number of values 'max' or less and stores to 'buf'.

API adds '¥0' to the end of the character string if the number of elements less than 'max'.

API returns a length of defined character string as return value (length excluding '¥0').

If the return value is bigger than 'max', all of the information is not stored.

It is possible to get a number of elements for numeric string without storing to 'buf' by setting '0' to 'max'.

Error 'E_NOEXS' is notified if system configuration information set by 'name' not defined.

If the system configuration information set by 'name' being by numeric string, API operates on the basis of character string, not transform to numeric string.

8.2.9 API for Domain Management Functions

Refer to ‘3.6 Domain Management Functions’ about the detail of domain management functions.

8.2.9.1 ts_sta_dmn – Start Domain

C Language Interface

[Safety API]	ER ercd = ts_sta_dmn(ID dmnid);
[Normal API]	None

Parameter

ID	dmnid	Domain ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_OBJ	Domain state Invalid (Domain is NOT on Domain stopped state (DORMANT))
E_DACV	Domain access protection violation (Target domain is System domain)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API starts to execute the domain set by 'dmnid'

API transits the initial task to ready state. The domain is transited to running state and then initial task on its domain is started to execute.

Request to start execution by this API is not queued. Error 'E_OBJ' is notified if this API issued for the domain other than on Domain stopped state.

8.2.9.2 ts_ext_dmn/tn_ext_dmn – Exit Domain

C Language Interface

[Safety API] void ts_ext_dmn(void);

[Normal API] void tn_ext_dmn(void);

Parameter

None

Return parameter

None

Error code

None (NOT returned to the context issuing this API)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API terminates the self domain normally and transits to Domain stopped state.

If the self domain transited to Domain stopped state, this API operates as follows.

- All tasks on the self domain are finished executing and deleted. The task issuing this API is operated equivalent to executing 'ts_exd_tsk/tn_exd_tsk'. Other tasks are finished executing forcibly and operated equivalent to deleting those tasks by executing 'ts_exd_tsk/tn_exd_tsk'.
- All kernel objects are deleted on the self domain (including a time event handler).
- All the devices opened by the task of the self domain are closed.

Exit of the domain is executed on the context of initial task in the self domain. If this API on operating, the domain is transited to state of dispatch disabled and the time event handler on the domain is s suppressed its execution.

If the API is issued in the portion other than task portion and quasi-task portion, error is detected and abnormal exception occurs.

8.2.9.3 ts_ter_dmn – Terminate Domain

C Language Interface

[Safety API]	ER ercd = ts_ter_dmn(ID dmnid);
[Normal API]	None

Parameter

ID	dmnid	Domain ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_OBJ	Object state Invalid (Domain is NOT on dormant state (DORMANT) or the self domain)
E_DACV	Domain access protection violation (Target domain is System domain)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API terminates the domain set by 'dmnid'.

API terminates the domain set by 'dmnid' and transits to Domain stopped state.

Termination of the domain is equivalent to exit the self domain by executing 'ts_ext_dmn/tn_ext_dmn'.
Termination of the domain is executed on the context of initial task in the self domain. If this API on operating, the domain is transited to state of dispatch disabled and the time event handler on the domain is suppressed its execution.
API finishes its execution before completing the exit operation for the domain in some cases. Program other than exit operation is not executed on the domain after issuing this API. It is possible to get the domain state by 'ts_ref_dmn/tn_ref_dmn'.

It is NOT possible to set the self domain itself. Error 'E_OBJ' is notified if setting.
It is NOT possible to terminate the system domain forcibly. Error 'E_DACV' is notified if setting.

8.2.9.4 ts_sus_dmn – Suspend Domain

C Language Interface

[Safety API]	ER ercd = ts_sus_dmn(ID dmnid);
[Normal API]	None

Parameter

ID	dmnid	Domain ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_OBJ	State of domain Invalid (Target domain is the self domain or system domain, or domain on prohibited suspending forcibly)
E_DACV	Domain access protection violation (Target domain is System domain)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API transits the domain set by 'dmnid' to suspended state and suspends to execute a task and an event handler on the self domain.

It is NOT possible to set the self domain itself. Error 'E_OBJ' is notified if setting.
It is NOT possible for system domain. In Safety domain, it is statically set by the setting at creating whether possible to suspend or not. Error'E_OBJ' is notified if setting the domain unable to be suspended.

Task and time event handler on suspended state in the self domain is out of the scheduling for executing a program on TRON Safe Kernel and not executed. State of the task and the time event handler keeps to the state at transiting to suspended state.
Domain on suspended state is released from its state by issuing 'ts_rsm_dmn'.

If issuing API repeatedly for the domain already on suspended state, API keeps its domain to be on suspended state and exits normally and then notifies 'E_OK'. Request of suspending is not queued. If issuing this API repeatedly by same domain, the domain is released from suspended state by one (1) execution of 'ts_rsm_dmn'.

8.2.9.5 ts_rsm_dmn – Resume Domain

C Language Interface

[Safety API]	ER ercd = ts_rsm_dmn(ID dmnid);
[Normal API]	None

Parameter

ID	dmnid	Domain ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_OBJ	State of domain Invalid (Target domain is NOT on suspended state)
E_DACV	Domain access protection violation (Target domain is System domain)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API releases the domain set by 'dmnid' from suspended state.

API releases the domain on suspended state by issuing 'ts_sus_dmn' from on its state and resumes the execution.

State of a task and a time event handler on the domain are kept on suspended state. The task and the time event handler are resumed their executions after releasing from on suspended state.

8.2.9.6 ts_get_oid/tn_get_oid – Get Object ID

C Language Interface

[Safety API]	ID oid = ts_get_oid(ID dmnid, UINT otype, CONST UB *oname);
[Normal API]	ID oid = tn_get_oid(ID dmnid, UINT otype, CONST UB *oname);

Parameter

ID	dmnid	Domain ID
UINT	otype	Type of kernel object
CONST UB	*oname	Kernel object name

Return parameter

ID	oid	Kernel object ID	
		or	Error code

Error code

E_ID	ID number Invalid ('dmnid' is invalid or NOT used
E_PAR	Parameter error (Kernel object type specified by ' otype ' is invalid, or ' oname' is invalid, or the length of string of 'oname' is '0' or more than '9')
E_OBJ	State of domain Invalid (Domain is on Dormant stopped state)
E_NOEXS	Object set by 'oname' NOT existed
E_DACV	Domain access protection violation (Set the other domain by Normal API or set System domain by Safety Domain by using Safety API)
E_MACV	Memory access violation (There is no access authority to 'oname'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API gets an ID of the kernel object set by 'otype' and 'oname' on the domain set by 'dmnid'.

Kernel object is set by the object type 'otype' and the object name 'oname'.

Object type is indicated as follows.

TN_TSK (0x01)	Task
TN_SEM (0x02)	Semaphore
TN_FLG (0x03)	Event flag
TN_MBF (0x05)	Message buffer
TN_MTX (0x07)	Mutex

TN_CYC (0x0a)	Cyclic handler
TN_ALM (0x0b)	Alarm handler

The self domain is set by setting '0' to 'dmnid (= DMN_SELF) '. It is permitted to set the self domain only in Normal API.

8.2.9.7 ts_inf_dmn/tn_inf_dmn – Reference Domain Statistics

C Language Interface

[Safety API] ER ercd = ts_inf_dmn(ID dmnid, T_IDMN *pk_idmn, BOOL clr);

[Normal API] ER ercd = tn_inf_dmn(ID dmnid, T_IDMN *pk_idmn, BOOL clr);

Parameter

ID	dmnid	Domain ID
T_IDMN*	pk_idmn	Pointer to the area returning statistical information of domain
BOOL	clr	Whether clearing statistical information of domain or not (Valid for Safety API)

Return parameter

ER	ercd	Error code
T_IDMN	*pk_idmn	Statistical information for domain

Statistical information for domain

```
typedef struct st_idmn {
        RELTIM   ctime;          /* Accumulated domain execution time (by milliseconds) */
        RELTIM   stime;          /* Consecutive domain execution time (by milliseconds) */
} T_IDMN;
```

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_PAR	Parameter error (Address set by 'pk_idmn' is invalid)
E_DACV	Domain access protection violation (Set the other domain by Normal API or set System domain by Safety domain by using Safety API)
E_MACV	Memory access violation (There is no access authority to 'pk_idmn'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API refers to statistical information for the domain set by 'dmnid'.

It is possible to set the self domain by 'dmnid=DMN_SELF(=0)'

It is possible to set the self domain only in Normal API. Error 'E_DACV' is notified if setting.

If System domain is set by Safety domain by using Safety API, error 'E_DACV' is notified.

'ctime' : means accumulated domain execution time. It is the accumulated time in executing the task and time event handler on the domain (by milliseconds). Its time is started to measure at executing the domain by 'ts_sta_dmn'. API adds the time executing the task and time event handler to the accumulated domain execution time , NOT the time executing other units such as an interrupt handler and so on. API adds the time executing the extended SVC called by the task.
The cases of resetting (clear by '0') 'ctime' are indicated as follows.

- Start to execute the domain by 'ts_sta_dmn'.
- Execute 'ts_inf_dmn/tn_inf_dmn' with 'clr=TRUE' (Reset after getting values).

'stime' : means consecutive domain execution time.
Consecutive domain execution time means the time executing the domain consecutively (by milliseconds). Therefore it means the time from starting to execute a task or a time event handler on the domain to changing to execute a task or a time event handler on other domain. Consecutive execution is not depended on the change of a task or a time event handler on the self domain.
Time executing extended SVC called by the target task is added, and time executing a task independent portion such as interrupt handler is excluded.
'stime' is reset (cleared to '0') in following case.
- A task or a time event handler changes to a task or a time event handler on other domain.

If 'clr=TRUE' in Safety API, accumulated domain execution time is reset (clear by '0') after getting statistical information. Setting of 'clr' is ignored in Normal API.

8.2.9.8 ts_ref_dmn/tn_ref_dmn – Reference Domain status

C Language Interface

[Safety API]	ER ercd = ts_ref_dmn(ID dmnid, T_RDMN *pk_rdmn);
[Normal API]	ER ercd = tn_ref_dmn(ID dmnid, T_RDMN *pk_rdmn);

Parameter

ID	dmnid	Domain ID
T_RDMN*	pk_rdmn	Pointer to the area returning a state of domain

Return parameter

ER	ercd	Error code
T_RDMN	rdmn	Domain state

State of domain

```
typedef     struct st_rdmn {
            ID          dmnno;              /* Domain number */
            UINT        dmnstat;            /* Domain state */
} T_RDMN;
```

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('dmnid' is invalid or NOT used)
E_PAR	Parameter error (Address set by 'pk_rdmn' is invalid)
E_DACV	Domain access protection violation (Set the other domain by Normal API or set System domain by Safety domain by using Safety API)
E_MACV	Memory access violation (There is no access authority to ‘pk_rdmn’. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API refers to state of the domain set by 'dmnid'.

It is possible to set the self domain by 'dmnid=DMN_SELF(=0)'.

It is possible to set the self domain only in Normal API. Error 'E_DACV' is notified if setting.

If System domain is set by Safety domain by using Safety API, error ‘E_DACV’ is notified.

'dmnno' : means a number of the self domain.

'dmnstat' : means domain state.

Domain state is indicated as follows.

TDS_RUN	0x00000001	Domain running state
TDS_WAI	0x00000008	Domain exit operation state
TDS_DMT	0x00000010	Domain stopped state
TDS_DISDSP	0x00000080	Dispatch disabled state
TDS_SUS	0x00000100	Domain forcibly stopped state

Dispatch disabled state(TDS_DISDSP) can be set together with only Domain running state (TDS_RUN). If a domain being on the state of running and dispatch disabled by 'ts_dis_dsp/tn_dis_dsp', its domain is set 'TDS_DISDSP'.

8.3 API for TRON Safe Kernel/SM

8.3.1 API for Subsystem Management Functions

Refer to ‘4.3 Subsystem Management Functions’ about the detail of the subsystem management functions.

8.3.1.1 ts_cal_svc/tn_cal_svc – Call SVC

C Language Interface

[Safety API]	ER ercd = ts_cal_svc(ID ssid, FN fncd, T_SVCPARA *pk_svcpara);
[Normal API]	ER ercd = tn_cal_svc(ID ssid, FN fncd, T_SVCPARA *pk_svcpara);

Parameter

ID	ssid	Subsystem ID
FN	fncd	Function code
T_SVCPARA	*pk_svcpara	Pointer to parameter to pass to extended SVC

Parameter to pass to extended SVC

```
typedef struct st_svcpara {
        UW      par[TS_MAX_SVCPARA];        Each parameter of extended SVC
} T_SVCPARA;
```

Return parameter

ER ercd Error code

or Return value of each extended SVC

Error code

E_OK	Normal completion
E_RSFN	Error for reserved function code number (‘ssid’ or ‘fncd’ is invalid)
E_PAR	Parameter error (Address set by 'pk_svcpara' is invalid)
E_DISWAI	Disabled call of extended SVC (‘TTX_SVC’ is set by ‘ts_dis_wai’ or ‘tn_dis_wai’)
E_DACV	Domain access protection violation (Not available in the domain of the caller of API)
E_MACV	Memory access violation (There is no access authority to ‘pk_svcpara’. Or there is no target memory area.)
Other	Other error code returned by each extended SVC

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

The API calls an extended SVC handler of subsystem specified by ‘syvid’.

Parameters of extended SVC shown by ‘fncd’ or ‘pk_svcpara’ are passed to the called extended SVC.

‘TS_MAX_SVCPARA’ is a number of implementation-defined which is 4 and overThe content of each member of ‘T_SVCPARA’ is defined in each subsystem.

If specified ‘ssid’ or fncd‘ is invalid, an error for reserved function code number (E_RSFN) is notified.

If address specified by ‘pk_svcpara’ is invalid, parameter error (E_PAR) is notified.

If the target extended SVC is not available from the domain to which execution program calling the API belongs, because of the attribute of the specified extended SVC, violation of Domain access protection (E_DACV) is notified.

In other case, an extended SVC handler is called and the return value becomes the one of the API.

[Supplement]

Generally each subsystem prepares the C language function or macro to call each extended SVC. The API is called by the C language or macro. Therefore the program using the extended SVC does not call the API directly. However it can call the API directly.

If ‘ssid’ or ‘fncd’ is invalid, parameter error (E_PAR) or idnumber error (E_ID) is not notified, but error for unsupport (E_RSFN) is notified, because it shows that the specified extended SVC does not exist in the system software. And if the C language function or macro described above is used, the program does not specify ‘ssid’ or ‘fncd’ directly, so E_RSFN is notified to avoid that the cause of error is unclear.

8.3.2 API for Device Management Functions

Refer to ‘4.1 Device Management Functions’ about the detail of the device management functions.

8.3.2.1 ts_opn_dev/tn_opn_dev – Open Device

C Language Interface

[Safety API]	ID dd = ts_opn_dev(CONST UB *devnm, UINT omode);
[Normal API]	ID dd = tn_opn_dev(CONST UB *devnm, UINT omode);

Parameter

CONST UB*	devnm	Device name
UINT	omode	Open mode

Return parameter

ID	dd	Device descriptor
	or	Error code

Error code

E_PAR	Parameter error (Address set by 'devmn' is invalid or 'omode' is invalid)
E_BUSY	Device on using (on opening exclusively)
E_NOEXS	Device NOT existed
E_LIMIT	Number of opening devices is beyond the maximum number possible to open
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation (Device NOT possible to open on this domain
E_MACV	Memory access violation (There is no access authority to 'devnm'. Or there is no target memory area.)
E_OBJ	Object state invalid (error occurs by creating semaphore)
Others	Error code returning from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API opens the device set by 'devnm' on the mode set by 'omode' and prepares for accessing the device.

API returns a device descriptor.

'omode': Set an open mode of device.

'omode' is set as follows.

omode := (TD_READ || TD_WRITE || TD_UPDATE) | [TD_EXCL || TD_WEXCL || TD_REXCL] | [TD_NOLOCK]

TD_READ	Read only

TD_WRITE	Write only
TD_UPDATE	Enable to read and write
TD_EXCL	Open exclusively (Prohibit concurrent opening)
TD_WEXC	Write exclusively (Prohibit concurrent opening by 'TD_WRITE' or 'TD_UPDATE')
TD_REXCL	Read exclusively (Prohibit concurrent opening by 'TD_READ' or 'TD_UPDATE')
TD_NOLOCK	Lock (Be resident) NOT needed

Open mode is defined in header file as follows.

```
#define TD_READ      (0x00000001U)     /* Read only */
#define TD_WRITE     (0x00000002U)     /* Write only */
#define TD_UPDATE    (0x00000003U)     /* Read and Write */
#define TD_EXCL      (0x00000100U)     /* Open exclusively */
#define TD_WEXCL     (0x00000200U)     /* Write exclusively */
#define TD_REXCL     (0x00000400U)     /* Read exclusively */
#define TD_NOLOCK    (0x00001000U)     /* Lock (Be resident) NOT needed */
```

The operation by opening the device repeatedly is depended on the combination of open modes. The combination of open modes is indicated in 'Table 8-4 Open mode of Device driver'.

Table 8-4 Open mode of Device driver

Present Open Mode		Concurrent Open Mode											
		No exclusive mode			TD_WEXCL			TD_REXCL			TD_EXCL		
		R	U	W	R	U	W	R	U	W	R	U	W
No exclusive mode	R	○	○	○	○	○	○	×	×	×	×	×	×
	U	○	○	○	×	×	×	×	×	×	×	×	×
	W	○	○	○	×	×	×	○	○	○	×	×	×
TD_WEXCL	R	○	×	×	○	×	×	×	×	×	×	×	×
	U	○	×	×	×	×	×	×	×	×	×	×	×
	W	○	×	×	×	×	×	○	×	×	×	×	×
TD_REXCL	R	×	×	○	×	×	○	×	×	×	×	×	×
	U	×	×	○	×	×	×	×	×	×	×	×	×
	W	×	×	○	×	×	×	×	×	○	×	×	×
TD_EXCL	R	×	×	×	×	×	×	×	×	×	×	×	×
	U	×	×	×	×	×	×	×	×	×	×	×	×
	W	×	×	×	×	×	×	×	×	×	×	×	×

R = TD_READ

W = TD_WRITE

U = TD_UPDATE

○ = Enable to open

× = Unable to open (E_BUSY)

It is possible to open the device either on Safety software or on Normal software. Devices are statically defined whether it opens on Safety software or on Normal software respectively.
The device which can be opened only by Safety software can be opened by 'ts_opn_dev' from software on Safety domain or System domain, and the device which can be opened only by Normal software can be opened by 'tn_opn_dev' from software on Normal domain.

API returns a device descriptor if executed normally. And then to read from and write to the device is executed by the device descriptor. The device descriptor belongs to the self domain of the opened task. Its device descriptor is not used on other domain.

If opening a physical device, API opens logical devices on the physical device on the basis of open exclusively, operates equivalent to their logical devices having the same open mode.

It is possible for Safety API to open the device which can be opned by Safety software, and for Normal API can open the device which can be opened by Normal software. Error 'E_DACV' is notified if it tries to open other.
This API creates a semaphore. Created semaphore is used for waiting for completion of request of close processing. If error occurs in creating semaphore, error 'E_OBJ' occurs. Created semaphore is deleted by 'ts_cls_dev/tn_cls_dev'.

8.3.2.2 ts_cls_dev/tn_cls_dev – Close Device

C Language Interface

[Safety API]	ER ercd = ts_cls_dev(ID dd, UINT option, TMO tmout);
[Normal API]	ER ercd = tn_cls_dev(ID dd, UINT option, TMO tmout);

Parameter

ID	dd	Device descriptor
UINT	option	Option for close
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error ('tmout' is invalid)
E_ID	'dd' Invalid or NOT opened
E_TMOUT	Time out error (Close processing is NOT completed within specified time)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API closes the device with the device descriptor set by 'dd'.

If the device on operating, API terminates its operation and closes the device.

'option' : Set an option for closing (Operation for closing the device).

'option' is send to the device driver controlling the device. Operation method by 'option' is defined on the specifications of the device driver.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if opened by the software on other domain.

This API specifies time out time to ‘tmout’.

If the processing time exceeds the time out time, error ‘E_TMOUT’ is returned and an abnormal exception occurs.

If Safety API specifies ‘TMO_FEVR’ or ‘TMO_POL’ as ‘tmout’, domain error ‘E_DOMAIN’ is notified.

If smaller than or equal to ‘-2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

[Supplement]

Device is also closed by other than this API as follows.

- All device descriptors on the domain are closed by its exit or termination ('ts_ext_dmn/tn_ext_dmn, ts_ter_dmn').

8.3.2.3 ts_rea_dev/tn_rea_dev – Start Read Device

C Language Interface

[Safety API]	ID reqid = ts_rea_dev(ID dd, W start, void *buf, W size, TMO tmout);
[Normal API]	ID reqid = tn_rea_dev(ID dd, W start, void *buf, W size, TMO tmout);

Parameter

ID	dd	Device descriptor
W	start	Starting position of reading (>=0: Inherent data, <0: Attributes data)
void*	buf	Buffer for storing a reading data
W	size	Size of reading data
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ID	reqid	Request ID
	or	Error code

Error code

E_ID	'dd' Invalid or NOT opened
E_PAR	Parameter error (Address set by 'buf' is invalid or Size set by 'size' is invalid or 'tmout' is invalid)
E_OACV	Open mode Invalid (Reading is NOT permitted)
E_LIMIT	Number of requests is beyond the upper limit
E_TMOUT	Fail on timeout (NOT received because of other request)
E_ABORT	Abort
E_CTX	Context error (Issued in the portionother than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'buf'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API starts to read inherent data or attributes data from the device set by 'dd'.

API notifies the request for reading to the device driver controlling the device. API returns request ID and exits, if the request is notified normally.

API starts to read and returns to the caller without checking the completion of reading. It is possible to check its completion of reading by 'ts_wai_dev/tn_wai_dev'. It shall NOT release 'buf' until completing reading.

If setting '0' and over to 'start', API reads an inherent data of the device. Unit of 'start' and 'size' for inherent data is defined on each device respectively.
If setting negative to 'start', API reads an attributes data of the device in attributes data number 'start' and size 'size' (by bytes). 'size' shall be bigger than the size of reading attributes data. It is NOT possible to read multiple attributes data at once. Error 'E_PAR' is notified if 'size' less than the size of reading attributes data.
If setting '0' to 'size', API checks a size possible to read currently, not reads.

If on reading or writing, whether receiving new request or not is depended on the device driver. If on the state unable to receive new request, the status transits to request waiting state in the processing of this API. 'tmout' is set the timeout time for the request waiting. If passing the timeout time, API notifies error 'E_TMOUT'. And then API cancels the request for reading from the device.
Domain error 'E_DOMAIN' is notified if 'TMO_FEVR' or 'TMO_POL' is set by Safety API.
If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if the device descriptor is opened by the software on other domain.

[Supplement]
API passes a request for reading from device to the device driver controlling its device. API finishes executing and returns to the caller after passing the request for reading to the device driver. Reading operation after that is executed by the device driver, therefore, its specification is defined by each device driver respectively.
Timeout time for API is the time until passing a request for reading to the device driver, and timeout time for the operation on the device driver is defined on each device driver respectively.

8.3.2.4 ts_wri_dev/tn_wri_dev – Start Write Device

C Language Interface

[Safety API]	ID reqid = ts_wri_dev(ID dd, W start, CONST void *buf, W size, TMO tmout);
[Normal API]	ID reqid = tn_wri_dev(ID dd, W start, CONST void *buf, W size, TMO tmout);

Parameter

ID	dd	Device descriptor
W	start	Starting position of writing (>=0: Inherent data, <0: Attribute data)
CONST void*	buf	Buffer for storing a writing data
W	size	Size of writing
TMO	tmout	Timeout time of request waiting (by milliseconds)

Return parameter

ID	reqid	Request ID
	or	Error code

Error code

E_ID	'dd' Invalid or NOT opened
E_PAR	Parameter error (Address set by 'buf' is invalid or Value of 'size' is invalid or 'tmout' is invalid)
E_OACV	Open mode Invalid (Writing is NOT permitted)
E_RONLY	Device NOT possible to write
E_LIMIT	Number of requests is beyond the upper limit
E_TMOUT	Fail on timeout (NOT received because of other request)
E_ABORT	Abort
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'buf'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API starts to write inherent data or attributes data to the device set by 'dd'.

API notifies the request for writing to the device driver controlling the device. API returns a request ID and exits, if notifies normally.

API starts to write and returns to the caller without checking the completion of writing. It is possible to check its completion of writing by 'ts_wai_dev/tn_wai_dev'. It shall NOT release 'buf' until completing writing.

If setting '0' and over to 'start', API writes an inherent data of the device. Unit of 'start' and 'size' for inherent data is defined on each device respectively.
If setting negative to 'start', API writes an attributes data of the device in attributes data number 'start' and size 'size' (by bytes). 'size' shall be equal to the size of writing attributes data. It is NOT possible to write multiple attributes data at once.
If setting '0' to 'size', API checks a size possible to write currently, not writes.

If on reading or writing, whether receiving new request or not is depended on the device driver. If on the state unable to receive new request, the status transits to request waiting state in the processing of this API. 'tmout' is set the timeout time for the request waiting. If passing the timeout time, API notifies error 'E_TMOUT'. And then API cancels the request for reading from the device.
Domain error ‘E_DOMAIN’ is notified if ‘TMO_FEVR’ or ‘TMO_POL’ is set by Safety API.
If smaller than or equal to ‘-2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if the device descriptor is opened by the software on other domain.

[Supplement]
API passes a request for writing to the device to the device driver controlling its device. API finishes executing and returns to the caller after passing the request for writing to the device driver. Actual writing operation to the device is executed by the device driver, and its specific processing is defined by each device driver respectively.
Timeout time for API is the time until passing a request for writing to the device driver, and timeout time for the operation on the device driver is defined on each device driver respectively.

8.3.2.5 ts_wai_dev/tn_wai_dev – Wait for Request Completion for Device

C language interface

[Safety API]	ID creqid = ts_wai_dev(ID dd, ID reqid, W *asize, ER *ioer, TMO tmout);
[Normal API]	ID creqid = tn_wai_dev(ID dd, ID reqid, W *asize, ER *ioer, TMO tmout);

Parameter

ID	dd	Device descriptor
ID	reqid	Request ID
W*	asize	Pointer to area where read/write size is returned
ER*	ioer	Pointer to area where input and output error are returned
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ID	creqid	Completed request ID
	or	Error code
W	asize	Actual read / write size
ER	ioer	Input and output error

Error code

E_ID	'dd' Invalid or NOT opened or 'reqid' Invalid or not the request to 'dd'
E_PAR	Parameter error (Address set by 'asize' or 'ioer' is invalid or 'tmout' is invalid)
E_OBJ	Request of 'reqid' waits for completion or cancel at the other task
E_NOEXS	No request of processing (only the case of 'reqid' ＝'0')
E_TMOUT	Fail on timeout (Processing ongoing)
E_ABORT	Abort processing
E_CTX	Context error (Issued in the state other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'asize' or 'ioer'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API waits for request completion of 'reqid' to device specified by 'dd'.

‘reqid’ is request ID indicating the request of the target and is returned by return value of ‘ts_rea_dev/tn_rea_dev’ or ‘ts_wri_dev/tn_wri_dev’.
If ‘reqid= 0’,this API waits for completion of one of the requests to device specified by ‘dd’. When this API is invoked, processing request is only the target of wait for completion. The processing requested after invoking this API is not the target of wait for completion.
If device receives plural requests, the order of completion of the request is not necessarily the order of request and it depends on device driver.

If the processing specified by ‘reqid’ is completed, this API exits normally. Then the size of the data read or written by the request is returned to ‘asize’. And the processing result from device driver (error code) is returned to ‘ioer’.

Timeout time of wait for completion is specified to ‘tmout’.
If processing time exceeds timeout time, error ‘E_TMOUT’ is returned. However even if processing time exceeds timeout time, requested processing continues. To check the completion of processing, this API needs to wait for completion again. If ‘reqid> 0’ and ‘tmout=TMO_FEVR(=-1)’, processing time does not exceed timeout time and the processing definitely completes. And if Safety API specifies ‘TMO_FEVR’, domain error ‘E_DOMAIN’ is notified. If processing time exceeds timeout time, the content of ‘asize’ is defined by the specifications of device driver respectively. The content of ‘ioer’ is invalid.
If smaller than or equal to ‘-2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

If wait for completion of request by this API cannot be executed correctly, error is returned. If error is returned to return value of this API, the contents of ‘asize’ and ‘ioer’ are invalid. And if error is returned to return valule, requested processing continues. So to check the completion of processing, this API needs to wait for completion again.

To wait for completion to the same request ID by plural tasks at the same time is impossible. If there is a task waiting for ‘reqid=0’, the other task cannot wait for completion to the same ‘dd’. Likewise if there is a task waiting for ‘reqid > 0’, the other task cannot wait for completion to ‘reqid=0’.

‘dd’ should be the descriptor of device opened by the software of the own domain. If it is the device descriptor opened by the other domain, error ‘E_DACV’ is notified.

8.3.2.6 ts_can_dev/tn_can_dev – Cancel Request of Device

C Language interface

[Safety API]	ID creqid = ts_can_dev(ID dd, ID reqid, TMO tmout);
[Normal API]	ID creqid = tn_can_dev(ID dd, ID reqid, TMO tmout);

Parameter

ID	dd	Device descriptor
ID	reqid	Request ID
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ID	creqid	Completed request ID
	or	Error code

Error code

E_ID	'dd' Invalid or NOT opened or 'reqid' Invalid or not the request to 'dd'
E_PAR	Parameter error ('tmout' is invalid)
E_TMOUT	Fail on timeout
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API cancels the request of 'reqid' to device specified by 'dd'.

'reqid' is request ID indicating the request of the target and is returned by return value of 'ts_rea_dev/tn_rea_dev' or 'ts_wri_dev/tn_wri_dev'.

Timeout time is specified to 'tmout'.

If processing time exceeds timeout time, error 'E_TMOUT' is returned and an abnormal exception occurs.

Domain error 'E_DOMAIN' is notified if 'TMO_FEVR' or 'TMO_POL' is set for Safety API.

If the other task waits for completion to the specified ID by 'ts_wai_dev/tn_wai_dev', the API exits by error 'E_ABORT'.

If smaller than or equal to '-2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the

value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if opened by the software on other domain.

[Supplement]

API requests the device driver to cancel its operation. Response time for the request by the device driver depends on its specifications, although operating for its request faster. Generally, timeout time, over the time until the device driver receiving a cancel request, is set. API notifies an abnormal exception at timeout because its timeout means the device driver have abnormal operation.

8.3.2.7 ts_srea_dev/tn_srea_dev – Synchronous Read

C Language Interface

[Safety API]	ER ercd = ts_srea_dev(ID dd, W start, void *buf, W size, W *asize, TMO tmout);
[Normal API]	ER ercd = tn_srea_dev(ID dd, W start, void *buf, W size, W *asize, TMO tmout);

Parameter

ID	dd	Device descriptor
W	start	Starting position of reading (>=0: Inherent data, <0: Attribute data)
void*	buf	Buffer for storing a reading data
W	size	Size of reading data
W*	asize	Pointer to the area returning a size of reading data
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ER	ercd	Error code
W	asize	Actual size of reading data

Error code

E_ID	'dd' Invalid or NOT opened
E_PAR	Parameter error (Address set by 'buf' is invalid or Size set by 'size' is invalid or 'tmout' is invalid)
E_TMOUT	Timeout error
E_OACV	Open mode Invalid (NOT permit reading)
E_LIMIT	Number of requests js beyond the upper limit
E_ABORT	Abort
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued indispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'buf'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API reads inherent data or attributes data from the device set by 'dd'.

API notifies the request for reading to the device driver controlling the device, and then transits to waiting state for the reading completion. API exits after the device driver finishing its operation.

Specification of 'start' and 'size' is equivalent to the one of 'ts_rea_dev/tn_rea_dev'.
Reading data is stored to 'buf'. Size of reading data is returned to 'asize'.

If an error occurrs in reading operation on the device driver, API returns the operation result (error code) from the device driver and then exits. Value of 'buf' and 'asize' is defined on the specifications of the device driver.

'tmout' : Set timeout time.
Error 'E_TMOUT' is notified if passing the timeout time. If reading request is not received on the device driver (on waiting state for the request), its request is canceled. If reading request has already been received, the operation of the device driver is suspended. This suspending is operated equivalent to the one of 'ts_can_dev/tn_can_dev'.
Domain error 'E_DOMAIN' is notified if 'TMO_FEVR' or 'TMO_POL' is set for Safety API.
If smaller than or equal to '−2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if opened by the software on other domain.

8.3.2.8 ts_swri_dev/tn_swri_dev – Synchronous Write

C Language Interface

[Safety API]	ER ercd = ts_swri_dev(ID dd, W start, CONST void *buf, W size, W *asize, TMO tmout);
[Normal API]	ER ercd = tn_swri_dev(ID dd, W start, CONST void *buf, W size, W *asize, TMO tmout);

Parameter

ID	dd	Device descriptor
W	start	Starting position of writing (>=0: Inherent data, <0: Attribute data)
CONST void*	buf	Buffer for storing a writing data
W	size	Size of writing data
W*	asize	Pointer to the area returning a size of writing data
TMO	tmout	Timeout time (by milliseconds)

Return parameter

ER	ercd	Error code
W	asize	Actual size of writing data

Error code

E_ID	'dd' Invalid or NOT opened
E_PAR	Parameter error (Address set by 'buf' or 'asize' is invalid or Size set by 'size' is invalid, or 'tmout' is invalid)
E_TMOUT	Timeout error
E_OACV	Open mode Invalid (NOT permit writing)
E_RONLY	Device NOT possible to write
E_LIMIT	Number of requests is beyond the upper limit
E_ABORT	Abort
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued indispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'buf'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API writes inherent data or attributes data to the device set by 'dd'.

API notifies the request for writing to the device driver controlling the device, and then transits to waiting state for the writing completion. API exits after the device driver finishing its operation.

Specification of 'start' and 'size' is equivalent to the one of 'ts_wri_dev/tn_wri_dev'.

Size of writing data is returned to 'asize'.

If an error occurrs in writing operation on the device driver, API returns the operation result (error code) from the device driver and then exits. Value of 'buf' and 'asize' is defined on the specifications of the device driver.

'tmout' : Set timeout time.

Error 'E_TMOUT' is notified if passing the timeout time. If writing request is not received on the device driver (on waiting state for the request), its request is canceled. If writng request has already been received, the operation of the device driver is suspended. This suspending is operated equivalent to the one of 'ts_can_dev/tn_can_dev'.

Domain error 'E_DOMAIN' is notified if 'TMO_FEVR' or 'TMO_POL' is set for Safety API.

If smaller than or equal to '−2' is set to 'tmout', parameter error 'E_PAR' is notified. And in case of Safety domain, if the value which is larger than 'Maximum timeout time' is set to 'tmout', parameter error 'E_PAR' is notified.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if opened by the software on other domain.

8.3.2.9 ts_evt_dev/tn_evt_dev – Send Drive Request Event to Device

C Language Interface

[Safety API]	INT retcode = ts_evt_dev(ID devid, INT evttyp, void *evtinf, TMO tmout);
[Normal API]	INT retcode = tn_evt_dev(ID devid, INT evttyp, void *evtinf, TMO tmout);

Parameter

ID	devid	Device ID of event destination for sending
INT	evttyp	Type of driver request event
void*	evtinf	Information for the event
TMO	tmout	Timeout time (by milliseconds)

Return parameter

INT	retcode	Return value from 'eventfn'
	or	Error code

Error code

E_ID	'devid' Invalid or Not available
E_NOEXS	Device set by 'devid' NOT existed
E_PAR	Parameter error (Address set by 'evtinf' is invalid, Value of 'evttyp' is invalid or 'tmout' is invalid)
E_TMOUT	Timeout error
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation (Not possible to access to the target device from this domain)
E_MACV	Memory access violation (There is no access authority to 'evtinf'. Or there is no target memory area.)
E_DOMAIN	Domain error
Others	Error code returned from a device driver

Valid Context

Task portion	Time event handler	Task independent portion
○	○	×

Description

API sends a driver request event to the device (to the device driver) set by 'devid'.

API exits after receiving a response for the driver request event.

'evttyp' : means a type of driver request event.

Functions (Operations) of driver request event and specification of 'evtinf' are defined on the basis of event type.

Operation time of the driver request event is defined on each device driver respectively.

'tmout' : Set timeout time.

Error 'E_TMOUT' and the abnormality exception are notified if passing the timeout time.

Domain error ‘E_DOMAIN’ is notified if ‘TMO_FEVR’ or ‘TMO_POL’ is set for Safety API.

If smaller than or equal to ‘−2’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified. And in case of Safety domain, if the value which is larger than ‘Maximum timeout time’ is set to ‘tmout’, parameter error ‘E_PAR’ is notified.

It is possible for Safety API to request to only the device possible to open by Safety software, and for Normal API the device possible to open by Normal software. Error 'E_DACV' is notified if tries to operate on other than those conditions.

[Supplement]

API requests the device driver to execute the operation by the driver request event. Response time for the request by the device driver depends on its specifications, although operating for its request faster. Generally, timeout time, over the time until the device driver receiving a cancel request, is set. API notifies an abnormal exception at timeout because its timeout means the device driver does not work normally.

8.3.2.10 ts_get_dev/tn_get_dev – Get Device Name

C Language Interface

[Safety API]	ID pdevid = ts_get_dev(ID devid, UB *devnm);
[Normal API]	ID pdevid = tn_get_dev(ID devid, UB *devnm);

Parameter

ID	devid	Device ID
UB*	devnm	Pointer to the area storing the device name

Return parameter

ID	pdevid	Device ID of physical device
	or	Error code
UB	devnm	Device name

Error code

E_ID	‘devid’ Invalid or Not available
E_NOEXS	Device set by 'devid' NOT existed
E_PAR	Parameter error (Address set by 'devnm' is invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation (Not possible to access to the target device from this domain)
E_MACV	Memory access violation (There is no access authority to ‘devnm’. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	×

Description

API gets a device name of the device set by 'devid' and stores to 'devnm'.

If 'devid' is for device ID of a physical device, its physical device name is stored in ‘devnm’.. If 'devid' for device ID of logical device, its logical device name is stored in ‘devnm’.
'devnm' is necessary to set 'L_DEVNM+1' bytes and over in length.
API returns a device ID of the physical device to which the 'devid' device belongs.

It is possible for Safety API to refer to only the device possible to open by Safety software, and for Normal API the device possible to open by Normal software. Error 'E_DACV' is notified if tries to operate on other than those conditions.

8.3.2.11 ts_ref_dev/tn_ref_dev – Get Device Information

C Language Interface

[Safety API]	ID devid = ts_ref_dev(CONST UB *devnm, T_RDEV *pk_rdev);
[Normal API]	ID devid = tn_ref_dev(CONST UB *devnm, T_RDEV *pk_rdev);

Parameter

CONST UB*	devnm	Device name
T_RDEV*	pk_rdev	Pointer to the area returning device information

Return parameter

ID	devid	Device ID
	or	Error code
T_RDEV	pk_rdev	Device information

Device information

```
typedef struct st_rdev {
        ATR     devatr;         /* Attributes of device */
        INT     blksz;          /* Block size of inherent data (-1 : undefined) */
        INT     nsub;           /* Number of sub-units */
        INT     subno;          /* 0 : Physical device    1 to 'nsub' : Sub-unit number + 1 */
} T_RDEV;
```

Error code

E_NOEXS	Device set by 'devid' NOT existed
E_PAR	Parameter error (Address set by 'devnm' or 'pk_rdev' is invalid)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or ininterrupt disabled state)
E_DACV	Domain access protection violation (Not possible to access to the target device from this domain)
E_MACV	Memory access violation (There is no access authority to ‘devnm’. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	×

Description

This API gets device information of the device set by devnm and stores to pk_rdev.

'devatr' : means a device attributes set at registering the device driver.

'blksz' : means a block size of inherent data (by bytes).

'nsub' : means a number of sub-units on a physical device for the device set by 'devnm'.

'subno' : means the sub-unit number. It is possible for 'subno' to set the value between '0' and 'nsub +1'.
For a physical device, 'subno' is set '0', and for a sub-unit, 'subno' is set '(sub-unit number + 1) = (1 to nsub + 1)'.

API has no device information if 'pk_rdev=NULL'.
API returns a device ID of the device set by 'devnm'.
It is possible for Safety API to refer to only the device possible to open by Safety software, and for Normal API the device possible to open by Normal software. Error 'E_DACV' is notified if it tries to operate other.

8.3.2.12 ts_oref_dev/tn_oref_dev –Get Device Information

C Language Interface

[Safety API]	ID devid = ts_oref_dev(ID dd, T_RDEV *rdev);
[Normal API]	ID devid = tn_oref_dev(ID dd, T_RDEV *rdev);

Parameter

ID	dd	Device descriptor
T_RDEV*	pk_rdev	Pointer to the area returning device information

Return parameter

ID	devid	Device ID
	or	Error code
T_RDEV	pk_rdev	Device information

Device information

```
typedef struct st_rdev {
        ATR     devatr;         /* Attributes of device */
        INT     blksz;          /* Block size of inherent data (-1 : undefined) */
        INT     nsub;           /* Number of sub-units */
        INT     subno;          /* 0 : Physical device    1 to 'nsub' : sub-unit number + 1 */
} T_RDEV;
```

Error code

E_ID	'dd' Invalid or NOT opened
E_PAR	Parameter error (Address set by 'rdev' is invalid
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion, or issued in dispatch disabled state or in interrupt disabled state)
E_DACV	Domain access protection violation ('dd' is opened on other domain)
E_MACV	Memory access violation (There is no access authority to 'rdev'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API gets device information for the device with the device descriptor set by 'dd' and stores its information to 'rdev'.

Specification of 'pk_rdev' is equivalent to the one of 'ts_ref_dev/tn_ref_dev'.

API has no device information if ' pk_rdev=NULL'.

API returns a device ID of the device set by 'dd' as a return value.

'dd' shall be the device descriptor for the device opened by the software on the self domain. Error 'E_DACV' is notified if opened by the software on other domain.

8.3.3 API for Interrupt Management Functions

Refer to ‘4.2 Interrupt Management Functions’ about the detail of the interrupt management functions.

8.3.3.1 ts_def_int – Define Interrupt Handler

C Language Interface

[Safety API]	ER ercd = ts_def_int(UINT dintno, CONST T_DINT *pk_dint);
[Normal API]	None

Parameter

UINT	dintno	Interrupt handler number
CONST T_DINT*	pk_dint	Pointer to the definition information for interrupt handler

Definition information for interrupt handler

```
typedef struct st_dint {
        ATR     intatr;         /* Attributes of interrupt handler */
        FP      inthdr;         /* Start-up address of an interrupt handler */
        INT     priority;       /* Execution priority */
        RELTIM  maxrtim;        /* Maximum consecutive execution time */
} T_DINT;
```

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_RSATR	Attributes reserved error ('intatr' is invalid or NOT used)
E_PAR	Parameter error ('dintno', 'pk_dint', 'inthdr' or 'maxrtim' are invalid or NOT available
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_MACV	Memory access violation (There is no access authority to 'pk_dint' or 'inthdr'. Or there is no target memory area.)
E_DOMAIN	Domain error

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

API defines an interrupt handler with the interrupt handler number set by 'dintno' and enables to use. API operates the mapping between the interrupt handler number set by 'dintno' and the address and attributes of the interrupt handler. Interrupt handler number is implementation-defined.

This API can be invoked by the task of System domain. If it is used by the other domain, error ‘E_DOMAIN’ is notified. And If it is used by other than task portion or quasi-task portion in System domain, error ‘E_CTX’ is notified.

'intatr' : Set an interrupt handler attributes.
To 'intatr', the following reservations are set.

intatr := [TIA_MLTINT]

TIA_MLTINT Permit multiple interrupts

```
#define TIA_MLTINT          (0x00000008)        /* Permit multiple interrupts */
```

Interrupt handler is coded as a function in C language described below. Pointer to its function '**inthdr**' is set to the start-up address for the interrupt handler 'inthdr'.

```
void        inthdr( UINT dintno )
{
            /*
                        Interrupt operations
             */
            return;             /* End of interrupt handler */
}
```

'dintno', passed to an interrupt handler, is the interrupt handler number on the interrupt. Its number is the same as the one set by this API.

'maxrtim' : Set maximum time possible to execute an interrupt handler consecutively by milliseconds..Minimum value of ‘maxrtim’ is 1 milisecond.
And maximum value of ‘maxrtim’ is the upper limit of consecutive execution time of system domain.
Abnormal exception is notified if an interrupt handler executing consecutively after passing the time of 'maxrtim'. API checks an execution time of the interrupt handler at its exit. The interrupt handler is not suspended on operating.

It is possible to call a system call after executing an interrupt handler.
Interrupt handler is executed as a task independent part. It is NOT possible to execute a system call transiting to waiting state or a system call setting the invoking task on the interrupt handler.
Dispatching (Changing an operating task) is not executed on executing the interrupt handler even if the task on running state (RUNNING) is transited to other state and other task to on running state (RUNNING) by issuing a system call on the interrupt handler. Interrupt handler is executed prior to its dispatching. And then the dispatching is executed after the interrupt handler finishing executing. Request for dispatching on executing an interrupt handler is not executed immediately and delayed to operate until the interrupt handler finishing executing. This is called 'Delay dispatching rule'.

If setting 'pk_dint=NULL', previous definition of the interrupt handler is released. The default handler is set on a state of releasing a definition for an interrupt handler.
The default handler is an interrupt handler defined by the system. It is available to use after initializing the kernel.

It is also possible to redefine the abnormal exception handler to the abnormal exception handler number which has already been defined.
It is not necessary to release the interrupt handler with the number before defining it again. Even if TRON Safe Kernel redefines a new handler to 'dintno' where an abnormal exception handler is already defined, an error does not occur.

[Supplement]

An interrupt handler can be registered by using Initial definition API at starting TRON Safe Kernel. This API is used to release or redefine the interrupt handler after starting TRON Safe Kernel.

8.3.4 API for Failure Diagnosis Functions

Refet to ‘6.3 Failure Diagnosis Functions’ about the detail of the failure diagnosis functions.

8.3.4.1 ts_sta_fdh – Start Failure Diagnosis Handler

C Language Interface

[Safety API]	ER ercd = ts_sta_fdh(ID fdhid);
[Normal API]	None

Parameter

ID	fdhid	Failure Diagnosis handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Exit Normally
E_ID	ID number Invalid ('fdhid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Failure diagnosis handler of 'fdhid' is NOT existed)
E_OACV	Object access violation (target failure diagnosis handler cannot operate API)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

API starts the work of failure diagnosis handler specified by 'fdhid' and makes the state of the handler 'Waiting state for starting'.

The failure diagnosis handler whose state is 'Waiting state for starting' is possible to execute by the specified period or the activation by API.

If the target attribute of the failure diagnosis handler is not one of the 'TA_API' (possible to operate by API), the API returns the error 'E_OACV'

If the state of the target failure diagnosis handler is already in 'Waiting state for starting', the handler does nothing and maintains the 'Waiting state for starting'.

8.3.4.2 ts_stp_fdh – Stop Failure Diagnosis Handler

C Language Interface

[Safety API]	ER ercd = ts_stp_fdh(ID fdhid);
[Normal API]	None

Parameter

ID	fdhid	Failure diagnosis handler ID

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('fdhid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Failure diagnosis handler of 'fdhid' is NOT existed)
E_OACV	Object access violation (target failure diagnosis handler cannot operate API)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

The API makes the state of failure diagnosis handler specified by 'fdhid' 'Stopped state'.

If the attribute of failure diagnosis handler is not 'TA_API' (possible to operate by API), error 'E_OACV' is notified.

If the target failure diagnosis handler is already in stopped state, this API does nothing.

8.3.4.3 ts_act_fdh – Activate Failure Diagnosis Handler

C Language Interface

[Safety API]	ER ercd = ts_act_fdh(ID fdhid);
[Normal API]	None

Parameter

ID	fdhid	Failure diagnosis ID

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_ID	ID number Invalid ('fdhid' is invalid or NOT used)
E_NOEXS	Object NOT existed (Failure diagnosis handler of 'fdhid' is NOT existed)
E_OBJ	Object state invalid (Target failure diagnosis handler is not 'Waiting state for starting')
E_OACV	Object access violation (target failure diagnosis handler cannot operate API)

Valid Context

Task portion	Time event handler	Task independent portion
○	○	○

Description

This API starts a failure diagnosis handler specified by 'fdhid'.

In particular this API transits the target failure diagnosis handler from 'Waiting state for starting' to 'Running state'.

If the target attribute of the failure diagnosis handler is not the one of 'TA_API' (possible to operate by API), the API returns the error 'E_OACV'.

If the state of the target failure diagnosis handler is not 'Waiting state for starting', the API returns the error 'E_OBJ'.

8.3.5 API for Abnormal Exception Functions

Refer to ‘6.2 Abnormal Exception Functions’ about the detail of the abnormal exception functions.

8.3.5.1 ts_def_aeh – Define Abnormal Exception Handler

C Language Interface

[Safety API]	ER ercd = ts_def_aeh(UINT aexpno, CONST T_DAEH* pk_daeh);
[Normal API]	None

Parameter

UINT	aexpno	Abnormal exception number
CONST T_DAEH*	pk_daeh	Pointer to abnormal exception definition information

Abnormal exception definition information

```
typedef struct st_daeh {
        ATR     aehatr;         /* Abnormal exception handler attribute */
        FP      aehdr;          /* Start-up address of abnormal exception handler */
} T_DAEH;
```

Return Parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_RSATR	Reserved attributes error ('aehatr' is invalid or NOT available)
E_PAR	Parameter error (value of 'aexpno' or 'pk_deah' or 'aehdr' are invalid or NOT available)
E_CTX	Context error (Issued in the portion other than a task portion and a quasi-task portion)
E_DACV	Domain access protection violation (executed by other than System domain)
E_MACV	Memory access violation (There is no access authority to 'pk_daeh' or 'aehdr'. Or there is no target memory area.)

Valid Context

Task portion	Time event handler	Task independent portion
○	×	×

Description

This API defines an abnormal exception handler to the abnormal exception handler number specified by 'aexpno'. Therefore this API executes the correspondence between the address and attributes of the abnormal exception handler number and the abnormal exception handler. An abnormal exception handler number is implementation-defined. (Refer to '6.2.2 Kinds of Abnormal exception' about the kinds of abnormal exception handler number)

This API can be issued by only the task of System domain. If it is used by the other domain, error 'E_DACV' is notified. And if it is used by other than task portion or quasi-task portion, error 'E_CTX' is notified.

An attribute of an abnormal exception handler to the target abnormal exception handler is specified to 'aehatr'. Following contents are specified to 'aehatr'.

aehatr := [TAA_UPDATE | TAA_RETURN]

TAA_UPDATE	Permit the change of an abnormal exception handler by Safety API
TAA_RETURN	Possible to return to original state after the completion of an abnormal exception handler

```
#define TAA_UPDATE     (0x00000001)     /* Permit the change of an abnormal exception handler by
                                           Safety API */
#define TAA_RETURN     (0x00000002)     /* Possible to return to original state after the completion
                                           of an abnormal exception handler */
```

An abnormal exception handler whose attribute is 'TAA_UPDATE' can be changed to the other abnormal exception handler by Safety API.
An abnormal exception handler whose attribute is not 'TAA_UPDATE' cannot be allowed to be changed to the other abnormal exception handler by this API.

An abnormal exception handler whose attribute is 'TAA_RETURN' can complete the handler by returning AE_RETURN(=1) as a return parameter and return to the original state. If it returns the 'AE_NORETURN'(=0 or other than the value of 'AE_RETURN') as a return parameter, TRON Safe Kernel transits to the safety state and stops the execution of all the programs.

An abnormal exception handler is described by the following format as a C language function. The following pointer to function 'aehdr' is specified to the start-up address of an abnormal exception handler.

```
INT      aehdr( UINT aexpno,   INT tskstat,   VP aexppar )
{
         /*
                  Abnormal exception processing
          */
         return AE_RETURN;            /* Return 'AE_RETURN' or 'AE_NORETURN' as a return
                                         parameter */
```

An abnormal exception handler number of occurred abnormal exception is passed to 'axepno' as a parameter of function. Kernel state when an abnormal exception occurs is passed to 'tskstat'. 'aexppar' is the information defined for each abnormal exception.
'aexpno' passed to an abnormal exception handler is an occurred abnormal exception handler number and same as the one specified by this API.

Invoking system call is possible in an abnormal exception handler.
However available API by an abnormal exception handler is restricted. Refer to ‘6.2.7 Available API by Abnormal exception handler’ in detail.

If an abnormal exception occurs, TRON Safe Kernel executes an abnormal exception handler in the highest priority. Then all the interrupts are prohibited. An abnormal exception handler should not permit the interrupt while executing

If ‘pk_aehdr=NULL’, this API release the definition of defined abnormal exception handler. Default handler is specified in the state that the definition of an abnormal exception handler is released.
Default handler is an abnormal exception handler defined by System. It is always available after Kernel initializing state.

It is also possible to redefine the abnormal exception handler to the abnormal exception handler number which has already been defined.
When TRON Safe Kernel redefines an abnormal exception handler, releasing the definition of the handler of the already defined number is not needed. Even If TRON Safe Kernel redefines a new handler to ‘aexpno’ where an abnormal exception handler is already defined, an error does not occur.

[Supplement]

An abnormal exception handler can be registered by using Initial definition API in starting TRON Safe Kernel. This API is used to release or redefine the interrupt handler after starting TRON Safe Kernel.

8.3.5.2 ts_ras_aexp – Raise Abnormal Exception

C Language interface

[Safety API]	ER ercd = ts_ras_aexp(UINT aexpno, VP aexppar);
[Normal API]	None

Parameter

UINT	aexpno	Abnormal exception number
VP	aexppar	Information of abnormal exception

Return parameter

ER	ercd	Error code

Error code

E_OK	Normal completion
E_PAR	Parameter error (value of ‘aexpno’ is invalid)
E_MACV	Memory access violation (There is no access authority to ‘aexppar’. Or there is no target memory area.)

Return Parameter

Task portion	Time event handler	Task independent portion
○	○	○

Description

The API generates the user-defined abnormal exception specified by ‘aexpno’.

If abnormal exception occurs, the execution of the other program is broken and corresponding abnormal exception handler is executed. The information specified by ‘aexppar’ is passed to the abnormal exception handler as a parameter.

If the abnormal exception handler corresponding to the user-defined abnormal exception generated is not defined, the error of undefined abnormal exception is notified.

9. Appendix

9.1 The background to designing the specification

This document contains the draft specification that has been reviewed and created by TRON Safe Kernel Working Group (April 2016－December 2017) aiming to design the specification of Real－time OS based on T－Kernel 2.0 Specification which supports the functional safety by TRON Forum.

The following are the names of the members who participated in TRON Safe Kernel WG. We express our appreciation of their cooperation.

TRON Safe Kernel WG member list (random order)

Chair: Yuichi Toyoyama, Hitachi ULSI Systems Co., Ltd.

Soichiro Kotaki, Hitachi ULSI Systems Co., Ltd.

Atsushi Tamaru, Hitachi ULSI Systems Co., Ltd.

Satoshi Oshima, Hitachi, Ltd.

Wakana Takeshita, Hitachi, Ltd.

Hiroshi Mine, Hitachi, Ltd.

Noboru Wakabayashi, Hitachi, Ltd.

Toshihiro Kawano, Renesas Electronics Corporation

Yuji Mouri, Renesas Electronics Corporation

Kenjiro Hayashi, eSOL Co., Ltd.

Yasuhiro Kobayashi, Socionext Inc.

Akira Matsui, Personal Media Corporation

Masato Kamio, Ubiquitous Computing Technology Corporation

Hiroyuki Yamada, Ubiquitous Computing Technology Corporation

Organizer: Shuji Yura, Ubiquitous Computing Technology Corporation