Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   - “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.
How to Use This Manual

Readers
This manual is intended for users who design and develop application system using RX MCU family.

Purpose
This manual is intended for users to understand the functions of real-time OS “RI600PX” manufactured by Renesas Electronics, described the organization listed below.

Organization
This manual can be broadly divided into the following units.

CHAPTER 1   OVERVIEW
CHAPTER 2   SYSTEM BUILDING
CHAPTER 3   MEMORY PROTECTION FUNCTIONS
CHAPTER 4   TASK MANAGEMENT FUNCTIONS
CHAPTER 5   TASK DEPENDENT SYNCHRONIZATION FUNCTIONS
CHAPTER 6   TASK EXCEPTION HANDLING FUNCTIONS
CHAPTER 7   SYNCHRONIZATION AND COMMUNICATION FUNCTIONS
CHAPTER 8   EXTENDED SYNCHRONIZATION AND COMMUNICATION FUNCTIONS
CHAPTER 9   MEMORY POOL MANAGEMENT FUNCTIONS
CHAPTER 10  TIME MANAGEMENT FUNCTIONS
CHAPTER 11  SYSTEM STATE MANAGEMENT FUNCTIONS
CHAPTER 12  INTERRUPT MANAGEMENT FUNCTIONS
CHAPTER 13  SYSTEM CONFIGURATION MANAGEMENT FUNCTIONS
CHAPTER 14  OBJECT RESET FUNCTIONS
CHAPTER 15  SYSTEM DOWN
CHAPTER 16  SCHEDULING FUNCTION
CHAPTER 17  SYSTEM INITIALIZATION
CHAPTER 18  DATA TYPES AND MACROS
CHAPTER 19  SERVICE CALLS
CHAPTER 20  SYSTEM CONFIGURATION FILE
CHAPTER 21  CONFIGURATOR cfg600px
CHAPTER 22  TABLE GENERATION UTILITY mkritblpx
APPENDIX A   WINDOW REFERENCE
APPENDIX B   FLOATING-POINT OPERATION FUNCTION
APPENDIX C   DSP FUNCTION
APPENDIX D   STACK SIZE ESTIMATION

How to Read This Manual
It is assumed that the readers of this manual have general knowledge in the fields of electrical engineering, logic circuits, microcomputers, C language, and assemblers.

To understand the hardware functions of the RX MCU
-> Refer to the User's Manual of each product.
**Conventions**

Data significance: Higher digits on the left and lower digits on the right

Note: Footnote for item marked with Note in the text

Caution: Information requiring particular attention

Remark: Supplementary information

Numeric representation: Decimal ... XXXX
Hexadecimal ... 0xXXXX

Prefixes indicating power of 2 (address space and memory capacity):

- K (kilo) $2^{10} = 1024$
- M (mega) $2^{20} = 1024^2$

**up4( data ):** A value in which data is rounded up to the multiple of 4.

**down( data ):** A integer part of data.
**Related Documents**

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI Series</td>
<td>Start R20UT0751E</td>
</tr>
<tr>
<td></td>
<td>Message R20UT0756E</td>
</tr>
<tr>
<td>RI600PX</td>
<td>Coding This document</td>
</tr>
<tr>
<td></td>
<td>Debug R20UT0950E</td>
</tr>
</tbody>
</table>

**Caution**

The related documents listed above are subject to change without notice. Be sure to use the latest edition of each document when designing.

All trademarks or registered trademarks in this document are the property of their respective owners.
TABLE OF CONTENTS

CHAPTER 1 OVERVIEW ... 14

1.1 Outline ... 14
   1.1.1 Real-time OS ... 14
   1.1.2 Multi-task OS ... 14
   1.1.3 Memory protection function ... 14

CHAPTER 2 SYSTEM BUILDING ... 15

2.1 Outline ... 15
2.2 Coding Processing Programs ... 16
2.3 Coding System Configuration File ... 16
2.4 Coding User-Own Coding Module ... 17
2.5 Creating Load Module ... 18
2.6 Build Options ... 23
   2.6.1 Service call information files and “-ri600_preinit_mrc” compiler option ... 23
   2.6.2 Compiler option for the boot processing file ... 24
   2.6.3 Kernel library ... 25
   2.6.4 Arrangement of section ... 26
   2.6.5 Initialized data section ... 29

CHAPTER 3 MEMORY PROTECTION FUNCTIONS ... 30

3.1 Outline ... 30
3.2 Domain, Memory object, Access permission vector ... 30
3.3 Restriction in the Number of Memory Objects ... 32
3.4 Trusted Domain ... 32
3.5 Change Access Permission ... 32
3.6 Protection of User Stack ... 32
3.7 Check Access Permission ... 33
3.8 Processor Mode ... 33
3.9 Enable MPU (Memory Protection Unit) ... 33
3.10 Access Exception Handler (_RI_sys_access_exception( ) ) ... 34
   3.10.1 User-Own Coding Module ... 34
3.11 Design of Memory Map ... 36
   3.11.1 The Restrictions regarding the Address of Memory Objects ... 36
   3.11.2 Area That Should Be the Inside of Memory Objects ... 36
   3.11.3 Area That Should Be the Outside of Memory Objects ... 37

CHAPTER 4 TASK MANAGEMENT FUNCTIONS ... 38

4.1 Outline ... 38
4.2 Tasks ... 38
   4.2.1 Task state ... 38
   4.2.2 Task priority ... 40
CHAPTER 8 EXTENDED SYNCHRONIZATION AND COMMUNICATION FUNCTIONS ... 118

8.1 Outline ... 118
8.2 Mutexes ... 118
  8.2.1 Priority inversion problem ... 119
  8.2.2 Current priority and base priority ... 119
  8.2.3 Simplified priority ceiling protocol ... 119
  8.2.4 Differences from semaphores ... 120
  8.2.5 Create mutex ... 121
  8.2.6 Delete mutex ... 122
  8.2.7 Lock mutex ... 123
  8.2.8 Unlock mutex ... 128
  8.2.9 Reference mutex state ... 129

Message Buffers 130
  8.3.1 Create message buffer ... 131
  8.3.2 Delete message buffer ... 132
  8.3.3 Send to message buffer ... 133
  8.3.4 Receive from message buffer ... 138
  8.3.5 Reference message buffer state ... 143
CHAPTER 9 MEMORY POOL MANAGEMENT FUNCTIONS ... 144

9.1 Outline ... 144

Fixed-Sized Memory Pools 145
  9.2.1 Create fixed-sized memory pool ... 145
  9.2.2 Delete fixed-sized memory pool ... 147
  9.2.3 Acquire fixed-sized memory block ... 148
  9.2.4 Release fixed-sized memory block ... 153
  9.2.5 Reference fixed-sized memory pool state ... 154

Variable-Sized Memory Pools 155
  9.3.1 Size of Variable-sized memory block ... 155
  9.3.2 Create variable-sized memory pool ... 156
  9.3.3 Delete variable-sized memory pool ... 157
  9.3.4 Acquire variable-sized memory block ... 158
  9.3.5 Release variable-sized memory block ... 163
  9.3.6 Reference variable-sized memory pool state ... 164

CHAPTER 10 TIME MANAGEMENT FUNCTIONS ... 165

10.1 Outline ... 165
10.2 System Time ... 165
  10.2.1 Base clock timer interrupt ... 165
  10.2.2 Base clock interval ... 165
10.3 Timer Operations ... 166
10.4 Delay Task ... 166
10.5 Time-out ... 166
10.6 Cyclic Handlers ... 167
  10.6.1 Basic form of cyclic handler ... 167
  10.6.2 Processing in cyclic handler ... 168
  10.6.3 Create cyclic handler ... 169
  10.6.4 Delete cyclic handler ... 170
  10.6.5 Start cyclic handler operation ... 171
  10.6.6 Stop cyclic handler operation ... 173
  10.6.7 Reference cyclic handler state ... 174
10.7 Alarm Handlers ... 175
  10.7.1 Basic form of alarm handler ... 175
  10.7.2 Processing in alarm handler ... 176
  10.7.3 Create alarm handler ... 177
  10.7.4 Delete alarm handler ... 178
  10.7.5 Start alarm handler operation ... 179
  10.7.6 Stop alarm handler operation ... 180
  10.7.7 Reference Alarm Handler State ... 181
10.8 System Time ... 182
  10.8.1 Set system time ... 182
  10.8.2 Reference system time ... 183
10.9 Base Clock Timer Initialization Routine (_RI_init_cmt_knl()) ... 184
  10.9.1 User-own cording module ... 184

CHAPTER 11 SYSTEM STATE MANAGEMENT FUNCTIONS ... 186
CHAPTER 12 INTERRUPT MANAGEMENT FUNCTIONS ... 196

12.1 Interrupt Type ... 196
12.2 Fast Interrupt of the RX-MCU ... 196
12.3 CPU Exception ... 196
12.4 Base Clock Timer Interrupt ... 197
12.5 Multiple Interrupts ... 197
12.6 Interrupt Handlers ... 198
   12.6.1 Basic form of interrupt handlers ... 198
   12.6.2 Register interrupt handler ... 199
12.7 Maskable Interrupt Acknowledgement Status in Processing Programs ... 199
12.8 Prohibit Maskable Interrupts ... 200
   12.8.1 Move to the CPU locked state by using loc_cpu, iloc_cpu ... 200
   12.8.2 Change PSW.IPL by using chg_ims, ichg_ims ... 200
   12.8.3 Change PSW.I and PSW.IPL directly (only for handlers) ... 200

CHAPTER 13 SYSTEM CONFIGURATION MANAGEMENT FUNCTIONS ... 201

13.1 Outline ... 201
13.2 Reference Version Information ... 201

CHAPTER 14 OBJECT RESET FUNCTIONS ... 202

14.1 Outline ... 202
14.2 Reset Data Queue ... 202
14.3 Reset Mailbox ... 203
14.4 Reset Message Buffer ... 204
14.5 Reset Fixed-sized Memory Pool ... 205
14.6 Reset Variable-sized Memory Pool ... 206

CHAPTER 15 SYSTEM DOWN ... 207

15.1 Outline ... 207
15.2 User-Own Coding Module ... 207
   15.2.1 System down routine (_RI_sys_dwn__( )) ... 207
   15.2.2 Parameters of system down routine ... 208

CHAPTER 16 SCHEDULING FUNCTION ... 212
16.1 Outline ... 212
16.2 Processing Unit and Precedence ... 212
16.3 Task Drive Method ... 212
16.4 Task Scheduling Method ... 213
  16.4.1 Ready queue ... 213
16.5 Task Scheduling Lock Function ... 214
16.6 Idling ... 215
16.7 Task Scheduling in Non-Tasks ... 215

CHAPTER 17 SYSTEM INITIALIZATION ... 216

17.1 Outline ... 216
17.2 Boot Processing File (User-Own Coding Module) ... 217
  17.2.1 Boot processing function (PowerON_Reset_PC()) ... 217
  17.2.2 Include kernel_ram.h and kernel_rom.h ... 218
  17.2.3 Compiler option for boot processing file ... 218
  17.2.4 Example of the boot processing file ... 219
17.3 Kernel Initialization Module (vsta_knl, ivsta_knl) ... 221
17.4 Section Initialization Function (_INITSCT()) ... 222
  17.4.1 Section information file (User-Own Coding Module) ... 223
17.5 Registers in Fixed Vector Table / Exception Vector Table ... 225

CHAPTER 18 DATA TYPES AND MACROS ... 226

18.1 Data Types ... 226
18.2 Constant macros ... 227
18.3 Function Macros ... 232
  18.3.1 Macros for Error Code ... 232
  18.3.2 Macros for Data Queue ... 232
  18.3.3 Macros for Fixed-sized Memory Pool ... 232
  18.3.4 Macros for Domain ... 232
  18.3.5 Access permission ... 233

CHAPTER 19 SERVICE CALLS ... 234

19.1 Outline ... 234
  19.1.1 Method for calling service calls ... 235
19.2 Explanation of Service Call ... 236

Task management functions 238
Task dependent synchronization functions 262
Task exception handling functions 280
Synchronization and communication functions (semaphores) 290
Synchronization and communication functions (eventflags) 303
Synchronization and communication functions (data queues) 319
Synchronization and communication functions (mailboxes) 339
Extended synchronization and communication functions (mutexes) 353
Extended synchronization and communication functions (message buffers) 368
Memory pool management functions (fixed-sized memory pools) 389
Memory pool management functions (variable-sized memory pools) 404
CHAPTER 20 SYSTEM CONFIGURATION FILE ... 481

20.1 Outline ... 481
20.2 Default System Configuration File ... 482
20.3 Configuration Information (static API) ... 482

System Information (system) 483
Note Concerning system.context 486
  20.5.1 Note concerning FPU and DSP ... 486
  20.5.2 Relationship with the compiler options “fint_register”, “base” and “pid” ... 488

Base Clock Interrupt Information (clock) 489
Maximum ID (maxdefine) 491
Domain Definition (domain[]) 497
Memory Object Definition (memory_object[]) 498
Task Information (task[]) 500
Semaphore Information (semaphore[]) 504
Eventflag Information (flag[]) 506
Data Queue Information (dataqueue[]) 508
Mailbox Information (mailbox[]) 510
Mutex Information (mutex[]) 512
Message Buffer Information (message_buffer[]) 513
Fixed-sized Memory Pool Information (memorypool[]) 515
Variable-sized Memory Pool Information (variable_memorypool[]) 517
Cyclic Handler Information (cyclic_hand[]) 519
Alarm Handler Information (alarm_handl[]) 522
Relocatable Vector Information (interrupt_vector[]) 524
Fixed Vector/Exception Vector Information (interrupt_fvector[]) 527
RAM Capacity Estimation 530
  20.23.1 BRI_RAM and RRI_RAM section ... 530
  20.23.2 BURI_HEAP section ... 533
  20.23.3 SURI_STACK section ... 533
  20.23.4 SI section ... 533

Description Examples 534

CHAPTER 21 CONFIGURATOR cfg600px ... 539

21.1 Outline ... 539
21.2 Start cfg600px ... 540
  21.2.1 Start cfg600px from command line ... 540
  21.2.2 Start cfg600px from CubeSuite+ ... 540

CHAPTER 22 TABLE GENERATION UTILITY mkritblpx ... 541
1.1 Outline

The RI600PX is a built-in real-time, multi-task OS that provides a highly efficient real-time, multi-task environment to increases the application range of processor control units. The RI600PX is a high-speed, compact OS capable of being stored in and run from the ROM of a target system. The RI600PX is based on the \( \mu \)ITRON4.0 specification. Furthermore, the RI600PX supports the memory protection function specified by \( \mu \)ITRON4.0 protection extension.

1.1.1 Real-time OS

Control equipment demands systems that can rapidly respond to events occurring both internal and external to the equipment. Conventional systems have utilized simple interrupt handling as a means of satisfying this demand. As control equipment has become more powerful, however, it has proved difficult for systems to satisfy these requirements by means of simple interrupt handling alone. In other words, the task of managing the order in which internal and external events are processed has become increasingly difficult as systems have increased in complexity and programs have become larger. Real-time OS has been designed to overcome this problem. The main purpose of a real-time OS is to respond to internal and external events rapidly and execute programs in the optimum order.

1.1.2 Multi-task OS

A “task” is the minimum unit in which a program can be executed by an OS. “Multi-task” is the name given to the mode of operation in which a single processor processes multiple tasks concurrently. Actually, the processor can handle no more than one program (instruction) at a time. But, by switching the processor’s attention to individual tasks on a regular basis (at a certain timing) it appears that the tasks are being processed simultaneously.

A multi-task OS enables the parallel processing of tasks by switching the tasks to be executed as determined by the system. One important purpose of a multi-task OS is to improve the throughput of the overall system through the parallel processing of multiple tasks.

1.1.3 Memory protection function

1) High-reliability system

To reduce a possibility of being unable to detect program glitches when debugging the program and causing a trouble in the market after the system has been shipped from the factory, this OS assures the system of high reliability. If memory data destruction occurs especially in a memory area in which the OS, etc. are stored, the system may produce a dangerous condition by, for example, operating erratically. However, since this OS is free of memory data corruptions, the system can continue operating normally, and is therefore assured of high system reliability.

2) Debug assistance

In systems without memory protection, a corruption of memory content by an illegal pointer behavior, etc. generally is not noticed until it actually comes to the surface as a trouble symptom. The cause of the trouble can only be identified by analyzing emulator’s trace data, which requires a large amount of time. The RI600PX can detect a bug when an illegal memory access is committed, enabling the debugging efficiency to be greatly increased.
CHAPTER 2 SYSTEM BUILDING

This chapter describes how to build a system (load module) that uses the functions provided by the RI600PX.

2.1 Outline

System building consists in the creation of a load module using the files (kernel library, etc.) installed on the user development environment (host machine) from the RI600PX's supply media.

Figure 2-1 shows the procedure of system building.

The RI600PX provides a sample program with the files necessary for generating a load module. The sample programs are stored in the following folder. The source files are stored in "appli" sub-folder.

<ri_sample> = <CubeSuite+_root>\SampleProjects\RX\device_name_RI600PX

- <CubeSuite+_root>
  Indicates the installation folder of CubeSuite+.
  The default folder is "C:\Program Files\Renesas Electronics\CubeSuite+\".
2.2 Coding Processing Programs

Code the processing that should be implemented in the system. In the RI600PX, the processing program is classified into the following five types, in accordance with the types and purposes of the processing that should be implemented.

- Tasks
  A task is processing program that is not executed unless it is explicitly manipulated via service calls provided by the RI600PX, unlike other processing programs (interrupt handler, cyclic handler and alarm handler).

- Task Exception Handling Routines
  When task exception is requested to a task, the task exception handling routine defined for the task is invoked. The requested exception code is passed to the task exception handling routine.

- Cyclic Handlers
  The cyclic handler is a routine started for every specified cycle time. The RI600PX handles the cyclic handler as a “non-task (module independent from tasks)”. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when a specified activation cycle has come, and the control is passed to the cyclic handler.

- Alarm Handlers
  The alarm handler is a routine started only once after the specified time. The RI600PX handles the alarm handler as a “non-task (module independent from tasks)”. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when a specified activation cycle has come, and the control is passed to the cyclic handler.

- Interrupt Handlers
  The interrupt handler is a routine started when an interrupt occurs. The RI600PX handles the interrupt handler as a “non-task (module independent from tasks)”. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when an interrupt occurs, and the control is passed to the interrupt handler.

Note For details about the processing programs, refer to “CHAPTER 4 TASK MANAGEMENT FUNCTIONS”, “CHAPTER 6 TASK EXCEPTION HANDLING FUNCTIONS”, “CHAPTER 10 TIME MANAGEMENT FUNCTIONS”, “CHAPTER 12 INTERRUPT MANAGEMENT FUNCTIONS”.

2.3 Coding System Configuration File

Code the SYSTEM CONFIGURATION FILE required for creating information files that contain data to be provided for the RI600PX.

Note For details about the system configuration file, refer to “CHAPTER 20 SYSTEM CONFIGURATION FILE”.

\( device\_name\): Indicates the device name which the sample is provided.

- SampleProjects
  Indicates the sample project folder of CubeSuite+.

- RX
  Indicates the sample project folder of RX MCU.

- \( device\_name\) RI600PX
  Indicates the sample project folder of the RI600PX. The project file is stored in this folder.
2.4 Coding User-Own Coding Module

- MEMORY PROTECTION FUNCTIONS
  - Access Exception Handler (_RI_sys_access_exception( ))
    The access exception handler will be invoked when a task or task exception handling routine accesses the memory that has not been permitted.

- TIME MANAGEMENT FUNCTIONS
  - Base Clock Timer Initialization Routine (_RI_init_cmt_knl( ))
    The base clock timer initialization routine is called at starting the RI600PX.

- SYSTEM DOWN
  - System down routine (_RI_sys_dwn__( ))
    The system down routine is called when the system down occurs.

- SYSTEM INITIALIZATION
  - Boot processing file
    For details, refer to “17.2 Boot Processing File (User-Own Coding Module)”.
  - Section information file (User-Own Coding Module)
    Informations for uninitialized data sections and initialized data sections are defined in the section information file.
2.5 Creating Load Module

Run a build on CubeSuite+ for files created in sections from "2.2 Coding Processing Programs" to "2.4 Coding User-Own Coding Module", and library files provided by the RI600PX and C compiler package, to create a load module.

1) Create or load a project
   Create a new project, or load an existing one.


2) Set a build target project
   When making settings for or running a build, set the active project.
   If there is no subproject, the project is always active.


3) Confirm the version
   Select the Realtime OS node on the project tree to open the Property panel.
   Confirm the version of RI600PX to be used in the [Kernel version] property on the [RI600PX] tab.

   Figure 2-2 Property Panel: [RI600PX] Tab
4) Set build target files
   For the project, add or remove build target files and update the dependencies.

   Note  See “CubeSuite+ Integrated Development Environment User’s Manual: RX Build” for details about adding
                  or removing build target files for the project and updating the dependencies.

The following lists the files required for creating a load module.

   - Source files created in “2.2 Coding Processing Programs”
        - Processing programs (tasks, cyclic handlers, alarm handlers, interrupt handlers)

   - System configuration file created in “2.3 Coding System Configuration File”
        - SYSTEM CONFIGURATION FILE

        Note  Specify “cfg” as the extension of the system configuration file name. If the extension is different, “cfg” is
                          automatically added (for example, if you designate “aaa.c” as a file name, the file is named as
                          “aaa.c.cfg”).

   - Source files created in “2.4 Coding User-Own Coding Module”
        - User-own coding module (system down routine, boot processing)

   - Library files provided by the RI600PX
        - Kernel library (refer to “2.6.3 Kernel library”)

   - Library files provided by the C compiler package
        - Standard library, runtime library, etc.

   Note 1  If the system configuration file is added to the Project Tree panel, the Realtime OS generated files node is
                  appeared. The following information files are appeared under the Realtime OS generated files node. However, these
                  files are not generated at this point in time.

              - System information header file (kernel_id.h)
              - Service call definition file (kernel_sysint.h)
              - ROM definition file (kernel_rom.h)
              - RAM definition file (kernel_ram.h)
              - System definition file (ri600.inc)
              - CMT timer definition file (ri_cmt.h)
              - Table file (ritable.src)
Figure 2-3  Project Tree Panel

Note 2  When replacing the system configuration file, first remove the added system configuration file from the project, then add another one again.

Note 3  Although it is possible to add more than one system configuration files to a project, only the first file added is enabled. Note that if you remove the enabled file from the project, the remaining additional files will not be enabled; you must therefore add them again.
5) Set the output of Realtime OS generation files
   Select the system configuration file on the project tree to open the Property panel.
   On the [System Configuration File Related Information] tab, set the output of realtime OS generation files, etc.

   Figure 2-4 Property Panel: [System Configuration File Related Information] Tab

6) Specify the output of a load module file
   Set the output of a load module file as the product of the build.

7) Set build options
   Set the options for the compiler, assembler, linker, and the like.
   Please be sure to refer to “2.6 Build Options”.
8) Run a build

Run a build to create a load module.


Figure 2-5  Project Tree Panel (After Running Build)

9) Save the project

Save the setting information of the project to the project file.

Note: See the "CubeSuite+ Integrated Development Environment User's Manual: Start" for details about saving the project.
2.6 Build Options

This section explains the build options that should be especially noted.

2.6.1 Service call information files and “-ri600_preinit_mrc” compiler option

The service call information file (mrc files) are generated to the same folder as object files at compilation of the source files that includes kernel.h file.

The name of service calls used in the source files are outputted in the mrc files. It is necessary to input all files to the table generation utility mkritblpx. If there is a leaking in the input file, service call modules that application uses might not be linked. In this case, the system down will occur when the service call is issued.

On the other hand, if the mrc files which are generated in the past and which is invalid in now are input to the mkritblpx, the service call modules that are not used in the application may be linked. In this case, there is no problem in the operation of the RI600PX but the module size uselessly grows.

Specify “-ri600_preinit_mrc” compiler option for the source file that includes kernel.h file even if this option is not specified, there is no problem in the operation of the RI600PX but the service call module that is not used in the application may be linked.

When application libraries are used, the mrc files that is generated at compilation of the library source should be inputted to the mkritblpx. If this way is difficult for you, make mrc file where name of using service calls is described (see belows), and input the mrc file to the mkritblpx.

Note, the system down will occur when the service call that is not linked is called.

```plaintext
sta_tsk
snd_mbx
rcv_mbx
prcv_mbx
```
2.6.2 Compiler option for the boot processing file

It is necessary to set "-nostuff" option for the boot processing file ("resetprg.c" in the sample project) like a mention in "17.2.3 Compiler option for boot processing file". If not, the RI600PX does not work correctly.

To set "-nostuff" option only for the boot processing file, please set any of the following in the [Individual Compile Options] tab of [Property] panel for the boot processing file. To set "-nostuff" option for all, please set any of the following in the [Compiler Options] tab of [Property] panel for [CC-RX (Build Tool)].

1) Set in the [Object] category
   Like Figure 2-6, set "Yes" in [Allocates uninitialized variables to 4-byte boundary alignment sections], [Allocates initialized variables to 4-byte boundary alignment sections] and [Allocates const qualified variables to 4-byte boundary alignment sections].

   ![Figure 2-6 [Object] category](image)

2) Set in the [Others] category
   Like Figure 2-7, add "-nostuff" to [Other additional options].

   ![Figure 2-7 [Others] category](image)
2.6.3 Kernel library

The kernel libraries are stored in the folders described in Table 2-1. Note, CubeSuite+ links the appropriate kernel library automatically, you need not consider the kernel libraries.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Compiler version corresponding to the library</th>
<th>Corresponding CPU core</th>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;ri_root&gt;\library\rxv1</td>
<td>RXv1 architecture</td>
<td>ri600lit.lib</td>
<td>For little endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ri600big.lib</td>
<td>For big endian</td>
</tr>
<tr>
<td>2</td>
<td>&lt;ri_root&gt;\library\rxv2</td>
<td>RXv1 architecture and RXv2 architecture</td>
<td>ri600lit.lib</td>
<td>For little endian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ri600big.lib</td>
<td>For big endian</td>
</tr>
</tbody>
</table>

Note 1  <ri_root> indicates the installation folder of RI600PX.
The default folder is "C:\Program Files\Renesas Electronics\CubeSuite+\RI600PX".

Note 2  The kernel described in item-2 of Table 2-1 is linked when compiler V2.01.00 or later is used. In the case of others, the kernel library described in item-1 of Table 2-1 is linked.
2.6.4 Arrangement of section

1) Naming convention of sections
In order to arrange sections to suitable memory objects, defining the naming convention of sections like the following examples is recommended.

- The 1st character: Section type
  - P: Program area
  - C: Constant area
  - B: Uninitialized data area
  - D: Initialized data area (ROM)
  - R: Variables area for initialized data (RAM) (generated by linker)
  - W: Switch statement branch table area (generated by compiler)
  - L: Literal area (generated by compiler)

- The 2nd character or subsequent ones
  - RI*: Reserved by the RI600PX
    This area is never accessed from user mode (= task context).
  - U*: Memory object or user stack
    This area is accessed from user mode (= task context).
  - S*: Excluding the above-mentioned
    This area is never accessed from user mode (= task context).
Table 2-2 shows RI600PX sections. The application must not use the RI600PX except SURI_STACK and BURI_HEAP.

<table>
<thead>
<tr>
<th>Section name</th>
<th>Attribute</th>
<th>Boundary alignment</th>
<th>ROM/RAM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRI_KERNEL CODE</td>
<td>CODE</td>
<td>1</td>
<td>ROM/RAM</td>
<td>RI600PX program</td>
</tr>
<tr>
<td>CRI_ROM</td>
<td>ROMDATA</td>
<td>4</td>
<td>ROM/RAM</td>
<td>RI600PX constant</td>
</tr>
<tr>
<td>DRI_ROM</td>
<td>ROMDATA</td>
<td>4</td>
<td>ROM/RAM</td>
<td>RI600PX initialized data (ROM)</td>
</tr>
<tr>
<td>FIX_INTERRUPT_VECTOR</td>
<td>ROMDATA</td>
<td>4</td>
<td>ROM</td>
<td>Fixed vector table/Exception vector table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Refer to “FIX_INTERRUPT_VECTOR section”</td>
</tr>
<tr>
<td>INTERRUPT_VECTOR</td>
<td>ROMDATA</td>
<td>4</td>
<td>ROM/RAM</td>
<td>Relocatable vector table (1kB)</td>
</tr>
<tr>
<td>SI</td>
<td>DATA</td>
<td>4</td>
<td>RAM</td>
<td>System stack</td>
</tr>
<tr>
<td>SURI_STACK</td>
<td>DATA</td>
<td>4</td>
<td>RAM</td>
<td>Default section for user stack area</td>
</tr>
<tr>
<td>BRI_RAM</td>
<td>DATA</td>
<td>4</td>
<td>RAM</td>
<td>- RI600PX variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Data queue area created by the system configuration file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Message buffer area created by the system configuration file (when the section is omitted)</td>
</tr>
<tr>
<td>RRI_RAM</td>
<td>DATA</td>
<td>4</td>
<td>RAM</td>
<td>RI600PX initialized data (RAM)</td>
</tr>
<tr>
<td>BURI_HEAP</td>
<td>DATA</td>
<td>4</td>
<td>RAM</td>
<td>- Fixed-sized memory pool area created by the system configuration file (when the section is omitted)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- variable-sized memory pool area created by the system configuration file (when the section is omitted)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Usually, this section should be arranged to memory object.</td>
</tr>
</tbody>
</table>
3 ) FIX_INTERRUPT_VECTOR section
The configurator cfg600px generates fixed vector table/exception vector table as FIX_INTERRUPT_VECTOR section according to the contents of definitions of “interrupt_fvector[]” in the system configuration file.

- At the time of RXv1 architecture use
  In the RXv1 architecture, fixed vector table is being fixed to address 0xFFFFFFF80. It is necessary to arrange the FIX_INTERRUPT_VECTOR section at address 0xFFFFFFF80.
  When the FIX_INTERRUPT_VECTOR section is not arranged to address 0xFFFFFFF80, RI600PX does not operate correctly.

- At the time of RXv2 architecture use
  In the RXv2 architecture, the name of fixed vector table is changed into exception vector table, and can set up the start address by EXTB register. The initial value of EXTB register at the time of reset is 0xFFFFFFF80, it is same as fixed interrupt vector table in RXv1 architecture.
  Usually, please arrange the FIX_INTERRUPT_VECTOR section to address 0xFFFFFFF80.
  When the FIX_INTERRUPT_VECTOR section is not arranged to address 0xFFFFFFF80, “interrupt_fvector[31]” (reset vector) in the system configuration file is ignored. Please generate the reset vector (address 0xFFFFFFFC) by the user side. And initialize EXTB register to the start address of FIX_INTERRUPT_VECTOR section in Boot processing function (PowerON_Reset_PC( ))

4 ) “aligned_section” linker option
The “aligned_section” linker option must be specified for following sections.

- The section specified as “memory_object[].start_address” in the system configuration file
- The section specified as “task[].stack_section” in the system configuration file
- SURI_STACK section

5 ) Attention concerning L and W section
The L section is literal area, and the W section is switch statement branch table area. These section are generated by the compiler.
The name of these section cannot be changed by using “#pragma section” directive.
Please note it as follows when a function executes as a task and the L and W section may be generated by compiling the source file.

A ) All functions in the source file executes only as tasks which belong to same domain
   Especially, there are no points of concern. The L and W section should be a memory object to be able to read from the domain.

B ) Functions in the source file executes as tasks which belong to separate domains
   Because a different name cannot be given to literal area and switch table area of each function, it is impossible to divide these area to separate memory object. Therefore, separate functions in order to the domain at running to individual source file and apply (A), or the L and W section should be a memory object to be able to read from all domain.
2.6.5 Initialized data section

About sections described in DTBL of the Section information file (User-Own Coding Module), it is necessary to perform setting to map sections placed on ROM to sections placed on RAM by using "-rom" linker option. Set [Link Options] tab of [Property] panel for [CC-RX (Build Tool)] like Figure 2-8.

![Figure 2-8 ROM to RAM mapped section](image)

Note In sample projects provided by RI600PX, it is already set up that the "DRI_ROM" section of RI600PX is mapped to "RRI_RAM" section.
CHAPTER 3 MEMORY PROTECTION FUNCTIONS

This chapter describes the memory protection functions performed by the RI600PX.

3.1 Outline

The RI600PX achieves following memory access protection function by using MPU (Memory Protection Unit) found in MCU. Note, handlers can access all address space.

1 ) Detection of illegal access by tasks and task exception handling routines
   Tasks and task exception handling routines can access only permitted memory objects. The access exception handler will be invoked if a task or task exception handling routine access the area that has not been permitted.

2 ) Protection of user stack area
   The user stack area of each task is inaccessible from other tasks. The access exception handler will be invoked if an user stack overflows or a task accesses an user stack for another task.

3 ) Detection of illegal access by the RI600PX
   Some service calls receives pointers as argument. The RI600PX inspects whether the invoking task can access to the area indicated by the pointer. The service call returns E_MACV error when the invoking task does not have the access permission to the area. And some service calls saves the context registers of the invoking task. If the user stack will overflow at the time, the service call returns E_MACV error.
   This feature is available only for service calls issued from task context.

3.2 Domain, Memory object, Access permission vector

The memory protection function is achieved by controlling the following.

- Who
- To where
- What access is permitted

The one that corresponds to “Who” is domain. Tasks and task exception handling routines belong to either of domain without fail. Domains are distinguished by domain ID with the value from 1 to 15. Domains is generated statically by the system configuration file.

The one that corresponds to “To where” is memory object, and the one that corresponds to “What access is permitted” is access permission vector.

Usually, memory objects are registered statically by the system configuration file. Memory objects can be registered dynamically by using ata_mem service call and unregistered by using det_mem service call. The start address of a memory object must be 16-bytes boundary, and the size must be multiple of 16.

Access permission vector represents whether tasks that belong to each domain can access (operand-read, operand-write, execution access) to the memory object.

The access exception handler will be invoked if a task accesses to the memory object that has not been permitted, or a task accesses other than memory objects and user stack for itself.

On the other hand, in handlers (interrupt handlers, cyclic handlers, alarm handlers and access exception handler), there is no concept of belonging domain. Handlers can access all address space.
Table 3-1 shows the operation concerning memory objects.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Static (system configuration file)</th>
<th>Dynamic (service call)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>memory_object[]</td>
<td>ata_mem</td>
</tr>
<tr>
<td>Unregistration</td>
<td>-</td>
<td>det_mem</td>
</tr>
<tr>
<td>Change access permission</td>
<td>-</td>
<td>sac_mem</td>
</tr>
<tr>
<td>Check access permission</td>
<td>-</td>
<td>vprb_mem</td>
</tr>
<tr>
<td>(Other than memory object can be checked.)</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Refer to state</td>
<td>-</td>
<td>ref_mem</td>
</tr>
</tbody>
</table>
3.3 Restriction in the Number of Memory Objects

The number of memory objects to which either of access (operand-read, operand-write and execution access) has been permitted by a certain domain is seven or less. Please design memory map with careful attention to this. The E_OACV error is detected when this restriction are no longer filled by ata_mem or sac_mem.

3.4 Trusted Domain

In the system which does not support protection mechanism other than memory protection, for example, the possibility of the following illegal accesses can be considered.

1 ) The task-A with malice does not have access permission for the memory object-M.
2 ) The coder wrote the source code for Task-A to create and start the task-B that belongs to the domain which has access permission for the memory object-M.
3 ) The illegal access is not detected if the task-B accesses the memory object-M.

To prevent such illegal access, the RI600PX supports “trusted domain”. The following service calls that gives the change to the composition of software can be called only from tasks that belongs to trusted domain. The E_OACV error is detected when a tasks that does not belong to trusted domain calls either of these service calls.

- cre_???, acre_???, del_???, def_tex, ata_mem, det_mem, sac_mem

3.5 Change Access Permission

The access permission for a memory object can be changed dynamically by using sac_mem. For instance, the following example showcases the requirement to change the access permission.

- In this example, it is required download application into memory that belongs to another domain and execute the application as a task .

1 ) Register the area for downloading as a memory object (ata_mem). At this time, specify to permit access the memory object from the domain-A that the task to download belongs to. Afterwards, download to the memory object area.
2 ) After downloading, set to be able to access the memory object from the domain-B (sac_mem). And create and start the downloaded code as the task that belongs to the domain-B.

3.6 Protection of User Stack

The user stack of each task can be accessed only by the task. The access exception handler will be invoked if the user stack overflows or a task accesses the user stack for another task.

When service call invoked from task uses user stack, the RI600PX inspects whether stack pointer points in the range of the user stack for invoking task. If not, the error E_MACV is returned.
3.7 Check Access Permission

In the program called from two or more domains, there is a scene that the program wants to judge whether the program can access the memory. In such a case, vprb_mem service call is useful. The vprb_mem inspects whether the specified task can do the specified access to the specified memory area.

3.8 Processor Mode

The memory protection by MPU (Memory Protection Unit) is effective only at user mode. In the RI600PX system, task context executes in user mode, and non-task context executes in supervisor mode.

3.9 Enable MPU (Memory Protection Unit)

The RI600PX enables MPU at initiation (vsta_knl, ivsta_knl). Do not disable MPU after the RI600PX has been initialized. If the MPU is disabled, the system operation cannot be guaranteed.
3.10 Access Exception Handler (_RI_sys_access_exception() )

The access exception handler will be invoked when a task or task exception handling routine accesses the memory that has not been permitted. For such situation, the access exception handler can either remove the factor of illegal access and return to normal program execution or be used for debug purposes.

3.10.1 User-Own Coding Module

The access exception handler must be implemented as user-own coding module.

Note: The source file for the access exception handler provided by the RI600PX as a sample file is “access_exc.c”.

- Basic form of access exception handler

The following shows the basic form of access exception handler.

```c
#include "kernel.h" // Provided by RI600PX
#include "kernel_id.h" // Generated by cfg600px

////////////////////////////////////////////////////////////
// Access exception handler
////////////////////////////////////////////////////////////
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);

void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
{
    ............
    ............
}
```

Note: The function name of access exception handler is “_RI_sys_access_exception”.

- Parameters

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameters</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UW pc;</td>
<td>R1</td>
<td>The instruction address that causes access exception</td>
</tr>
<tr>
<td>I</td>
<td>UW psw;</td>
<td>R2</td>
<td>The PSW value at access exception</td>
</tr>
<tr>
<td>I</td>
<td>UW sts;</td>
<td>R3</td>
<td>Factor of access exception Value of MPESTS register in the MPU (Memory Protection Unit) is passed.</td>
</tr>
<tr>
<td>I</td>
<td>UW addr;</td>
<td>R4</td>
<td>In the case of operand access error, the accessed address ( = Value of MPDEA register in the MPU) is passed. In the case of execution access error, the addr is indeterminate.</td>
</tr>
</tbody>
</table>

- Stack

The access exception handler uses the system stack.

- Service call

The access exception handler can issue service calls whose “Useful range” is “Non-task”.

```c
```
- PSW register when processing is started

Table 3-2  PSW Register When Access Exception Handler is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>All interrupts are masked.</td>
</tr>
<tr>
<td>IPL</td>
<td>Same before exception</td>
<td>Do not lower IPL more than the start of pro-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cessing.</td>
</tr>
<tr>
<td>PM</td>
<td>0</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>System stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
3.11 Design of Memory Map

This section explains information required for a design of memory map.

3.11.1 The Restrictions regarding the Address of Memory Objects

The start address of memory objects must be 16-bytes boundary, and the size must be multiple of 16. When a memory object is registered by the static API “memory_object[]” in the system configuration file, the start address can be specified by absolute address value or section name. To specify I/O register area, absolute address value should be used. When specifying section name, the start section and end section of the memory object should be specified. In this case, the sections must be arranged as assumption at linking. For example, specify “aligned_section” linker-option for the section specified for “start of memory object” (memory_object[].start_address) because the start address of memory object must be 16-bytes boundary. And the size of memory objects must be multiple of 16. In other words, the termination address of memory object must be multiple of 16 + 15. But, the end section (specified for memory_object[].end_address) does not necessary become just like that. When the termination address of the end section is not multiple of 16 + 15, the area from the termination address + 1 to next multiple of 16 + 15 is also treated with a part of the memory object. Therefore, don’t arrange any sections in the range from the termination address + 1 to next multiple of 16 + 15.

- Example
  When specifying “CU_DOM1” for “memory_object[].end_address” and the termination address of CU_DOM1 section is 0xFFFF1003, do not arrange any sections from 0xFFFF1004 to 0xFFFF100F. To achieve this requirement, specify “aligned_section” linker option to the section which follows “CU_DOM1”.

3.11.2 Area That Should Be the Inside of Memory Objects

1) Area that is accessed by tasks
   Tasks can access only memory objects to which the permission is appropriately set except it's own user stack. Therefore, it is necessary that program sections, constant sections, uninitialized data sections and initialized data sections accessed by tasks should be allocated to the inside of memory objects. Moreover, when the task accesses I/O area, the I/O area should be the inside of memory objects.

2) Message area handled by mailbox
   The message must be generated in the memory objects that both transmitting task and receiving task can access. However, the management table exists in the top of message area. The system operation cannot be guaranteed if the management table is destroyed. For this reason, data queue or message buffer is recommended for message communication.

3) Fixed-sized and variable-sized memory pool area
   The memory pool area should be the inside of memory object which can be accessed by tasks that use memory blocks. However, the RI600PX generates management tables in the memory pool area. The system operation cannot be guaranteed if the management table is destroyed.

   - A fixed-sized memory pool is created by the system configuration file
     The fixed-sized memory pool area is generated in the section indicated by “memorypool[].section”. When “memorypool[].section” is omitted, the fixed-sized memory pool area is generated in the “BURI_HEAP” section.

   - A fixed-sized memory pool is created by cre_mpf or acre_mpf
     Application should acquire the fixed-sized memory pool area, and specify the start address for cre_mpf or acre_mpf.

   - A variable-sized memory pool is created by the system configuration file
     The variable-sized memory pool area is generated in the section indicated by “variable_memorypool[].mpl_section”. When “variable_memorypool[].mpl_section” is omitted, the variable-sized memory pool area is generated in the “BURI_HEAP” section.

   - A variable-sized memory pool is created by cre_mpl or acre_mpl
     Application should acquire the variable-sized memory pool area, and specify the start address for cre_mpl or acre_mpl.
3.11.3 Area That Should Be the Outside of Memory Objects

1 ) All the RI600PX sections except BURI_HEAP
   The RI600PX sections except BURI_HEAP should be allocated to the outside of memory objects because these sections are accessed only by the RI600PX. Refer to “2.6.4 Arrangement of section” for the RI600PX sections.

2 ) User stack area for tasks
   The user stack area for tasks should be outside of memory objects. The correct system operation cannot be guaranteed if the user stack area overwraps with either all user stacks and memory objects.
   - A task is created by the system configuration file
     The user stack area is generated in the section indicated by “task[]_stack_section”. When “task[]_stack_section” is omitted, the user stack area is generated in the “SURI_STACK” section.
   - A task is created by cre_tsk or acre_tsk
     Application should acquire the user stack area, and specify the start address for cre_tsk or acre_tsk.

3 ) Data queue area
   The data queue area should be outside of memory objects. The correct system operation cannot be guaranteed if the data queue area overwraps with either all user stacks and memory objects.
   - A data queue is created by the system configuration file
     The data queue area is generated in the “BRI_RAM” section of the RI600PX.
   - A data queue is created by cre_dtq or acre_dtq
     Application should acquire the data queue area, and specify the start address for cre_dtq or acre_dtq.

4 ) Message buffer area
   The message buffer area should be outside of memory objects. The correct system operation cannot be guaranteed if the message buffer area overwraps with either all user stacks and memory objects.
   - A message buffer is created by the system configuration file
     The message buffer area is generated in the “BRI_RAM” section of the RI600PX.
   - A message buffer is created by cre_mbf or acre_mbf
     Application should acquire the message buffer area, and specify the start address for cre_mbf or acre_mbf.

5 ) Fixed-sized memory pool management area
   The fixed-sized memory pool management area should be outside of memory objects. The correct system operation cannot be guaranteed if the fixed-sized memory pool management area overwraps with either all user stacks and memory objects.
   - A fixed-sized memory pool is created by the system configuration file
     The data queue area is generated in the “BRI_RAM” section of the RI600PX.
   - A fixed-sized memory pool is created by cre_mpf or acre_mpf
     Application should acquire the fixed-sized memory pool management area, and specify the start address for cre_mpf or acre_mpf.
CHAPTER 4 TASK MANAGEMENT FUNCTIONS

This chapter describes the task management functions performed by the RI600PX.

4.1 Outline

The task management functions provided by the RI600PX include a function to reference task statuses such as priorities and detailed task information, in addition to a function to manipulate task statuses such as generation, activation and termination of tasks.

4.2 Tasks

A task is processing program that is not executed unless it is explicitly manipulated via service calls provided by the RI600PX, unlike other processing programs (interrupt handler, cyclic handler and alarm handler), and is called from the scheduler.

Note The execution environment information required for a task’s execution is called “task context”. During task execution switching, the task context of the task currently under execution by the RI600PX is saved and the task context of the next task to be executed is loaded.

4.2.1 Task state

Tasks enter various states according to the acquisition status for the OS resources required for task execution and the occurrence/non-occurrence of various events. In this process, the current state of each task must be checked and managed by the RI600PX. The RI600PX classifies task states into the following seven types.

Figure 4-1 Task State
1) Non-existent state
   The task has not been registered in the RI600PX. This is a virtual state.

2) DORMANT state
   State of a task that is not active, or the state entered by a task whose processing has ended.
   A task in the DORMANT state, while being under management of the RI600PX, is not subject to RI600PX scheduling.

3) READY state
   State of a task for which the preparations required for processing execution have been completed, but since another task with a higher priority level or a task with the same priority level is currently being processed, the task is waiting to be given the CPU's use right.

4) RUNNING state
   State of a task that has acquired the CPU use right and is currently being processed.
   Only one task can be in the running state at one time in the entire system.

5) WAITING state
   State in which processing execution has been suspended because conditions required for execution are not satisfied.
   Resumption of processing from the WAITING state starts from the point where the processing execution was suspended.
   The value of information required for resumption (such as task context) immediately before suspension is therefore restored.
   In the RI600PX, the WAITING state is classified into the following 12 types according to their required conditions and managed.

<table>
<thead>
<tr>
<th>Table 4-1 WAITING State</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAITING State</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Sleeping state</td>
</tr>
<tr>
<td>Delayed state</td>
</tr>
<tr>
<td>WAITING state for a semaphore resource</td>
</tr>
<tr>
<td>WAITING state for an eventflag</td>
</tr>
<tr>
<td>Sending WAITING state for a data queue</td>
</tr>
<tr>
<td>Receiving WAITING state for a data queue</td>
</tr>
<tr>
<td>Receiving WAITING state for a mailbox</td>
</tr>
<tr>
<td>WAITING state for a mutex</td>
</tr>
<tr>
<td>Sending WAITING state for a message buffer</td>
</tr>
<tr>
<td>Receiving WAITING state for a message buffer</td>
</tr>
<tr>
<td>WAITING state for a fixed-sized memory block</td>
</tr>
<tr>
<td>WAITING state for a variable-sized memory block</td>
</tr>
</tbody>
</table>

6) SUSPENDED state
   State in which processing execution has been suspended forcibly.
   Resumption of processing from the SUSPENDED state starts from the point where the processing execution was suspended.
   The value of information required for resumption (such as task context) immediately before suspension is therefore restored.

7) WAITING-SUSPENDED state
   State in which the WAITING and SUSPENDED states are combined.
   A task enters the SUSPENDED state when the WAITING state is cancelled, or enters the WAITING state when the SUSPENDED state is cancelled.
4.2.2 Task priority

A priority level that determines the order in which that task will be processed in relation to the other tasks is assigned to each task.

As a result, in the RI600PX, the task that has the highest priority level of all the tasks that have entered an executable state (RUNNING state or READY state) is selected and given the CPU use right.

In the RI600PX, the following two types of priorities are used for management purposes.

- Current priority
  The RI600PX performs the following processing according to current priority.
  - Task scheduling (Refer to “16.4 Task Scheduling Method”)
  - Queuing tasks to a wait queue in the order of priority

Note The current priority immediately after it moves from the DORMANT state to the READY state is specified at creating the task.

- Base priority
  Unless mutex is used, the base priority is the same as the current priority. When using mutex, refer to “8.2.2 Current priority and base priority”.

Note 1 In the RI600PX, a task having a smaller priority number is given a higher priority.

Note 2 The priority range that can be specified in a system can be defined by Maximum task priority (priority) in System Information (system) when creating a system configuration file.
4.2.3 Basic form of tasks

The following shows the basic form of tasks.

```c
#include "kernel.h"              /* Standard header file definition */
#include "kernel_id.h"           /* Header file generated by cfg600px */  
#pragma task
Task1
/* Refer to note 4 */
void Task1 (VP_INT exinf);          /* Refer to note 4 */
void Task1 (VP_INT exinf)
{
    /* ........ */
    ext_tsk ();                     /* Terminate invoking task */
}
```

**Note 1** The following information is passed to `exinf`.

<table>
<thead>
<tr>
<th>How to activate</th>
<th><code>exinf</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_ACT attribute is specified at the task creation</td>
<td>Extended information specified at the task creation.</td>
</tr>
<tr>
<td><code>act_tsk</code> or <code>iat_tsk</code></td>
<td></td>
</tr>
<tr>
<td><code>sta_tsk</code> or <code>ista_tsk</code></td>
<td>Start code (<code>stacd</code>) specified by <code>sta_tsk</code> or <code>ista_tsk</code></td>
</tr>
</tbody>
</table>

**Note 2** When the return instruction is issued in a task, the same processing as `ext_tsk` is performed.

**Note 3** For details about the extended information, refer to “4.5 Activate Task”.

**Note 4** These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.
4.2.4 Internal processing of task

In the RI600PX, original dispatch processing (task scheduling) is executed during task switching. Therefore, note the following points when coding tasks.

- **Stack**
  Tasks use user stacks that are defined in Task Information (task[]), cre_tsk or acre_tsk.

- **Service call**
  Tasks can issue service calls whose “Useful range” is “Task”.

- **PSW register when processing is started**

  Table 4-2  PSW Register When Task Processing is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td>All interrupts are acceptable.</td>
</tr>
<tr>
<td>IPL</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>PM</td>
<td>1</td>
<td>User mode</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>User stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **FPSW register when processing is started**

  When setting of Task context register (context) in System Information (system) includes “FPSW”, the FPSW when processing is started is shown in Table 4-3. The FPSW when processing is undefined in other cases.

  Table 4-3  FPSW Register When Task Processing is Started

<table>
<thead>
<tr>
<th>Compiler options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-round</td>
<td>-denormalize</td>
</tr>
<tr>
<td>nearest (default)</td>
<td>off (default)</td>
</tr>
<tr>
<td></td>
<td>0x00000100 (Only DN bit is 1.)</td>
</tr>
<tr>
<td></td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>zero</td>
<td>off (default)</td>
</tr>
<tr>
<td></td>
<td>0x00000101 (Only DN bit and RM bit are 1.)</td>
</tr>
<tr>
<td></td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>1 (Only RM bit is 1.)</td>
</tr>
</tbody>
</table>

4.2.5 Processor mode of task

The processor mode at the time of task execution is always user mode. It is impossible to execute a task in the supervisor mode. Processing to execute in the supervisor mode should be implemented as an interrupt handler for INT instruction. For example, the WAIT instruction, that changes the CPU to the power saving mode, is privilege instruction. The WAIT instruction should execute in the supervisor mode.

Note, INT #1 to #8 are reserved by the RI600PX, application cannot use INT #1 to #8.
4.3 Create Task

Tasks are created by one of the following methods.

1) Creation by the system configuration file
   The static API “task[]” described in the system configuration file creates a task.
   Refer to “20.10 Task Information (task[])” for the details of “task[]”.

2) Creation by cre_tsk or acre_tsk
   The cre_tsk creates a task with task ID indicated by parameter tskid according to the content of parameter pk_ctsk.
   The acre_tsk creates a task according to the content of parameter pk_ctsk, and returns the created task ID.
   The information specified is shown below.

   - Task attribute (tskatr)
     The following informations are specified as tskatr.
     - The domain to which the task belongs
       Refer to “3.2 Domain, Memory object, Access permission vector” for the details of the domain.
     - Specification started after creation (TA_ACT attribute)
       When the TA_ACT attribute is specified, the created task makes a transition to the READY state. When
       the TA_ACT attribute is not specified, the created task makes a transition to the DORMANT state.

   - Extended information (exinf)
   - Task start address (task)
   - Task initial priority (itskpri)

   - User stack size (stksz), Start address of user stack (stk)
     The user stack area must satisfy the following.
     - The start address must be 16-bytes boundary.
     - The size must be multiple of 16.
     - The user stack area must not overwrap with either all user stacks and all memory objects.

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_tsk as a representative.
Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include    "kernel.h"               /*Standard header file definition*/
#include    "kernel_id.h"            /*Header file generated by cfg600px*/

extern void Task2 (VP_INT exinf);
#include    "kernel_id.h"            /*Header file generated by cfg600px*/
#pragma section B SU_STACK2         /*Section for user stack area*/
    /*It is necessary to locate 16-byte boundary*/
    /*address at linking.*/
#define STKSZ2 256                      /*The user stack size must be multiple of 16.*/
static UW Stack2[STKSZ2/sizeof(UW)]; /*User stack area*/
#pragma section

#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);         /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ER_ID  tskid;                           /*Declares variable*/
    T_CTSK pk_ctsk = {                     /*Declares and initializes variable*/
        TA_DOM(1)|TA_ACT,                  /*Task attribute (tskatr)*/
        0,                                 /*Extended information (exinf)*/
        (FP) Task2,                        /*Task start address (task)*/
        1,                                 /*Task initial priority (itskpri)*/
        STKSZ2,                            /*User stack size (stksz)*/
        (VP) Stack2                        /*Start address of user stack (stk)*/
    };
    /* ......... */
    tskid = acre_tsk ( &pk_ctsk );        /*Creates task*/
    /* ......... */
}
```
4.4 Delete Task

Tasks are deleted by one of the following methods.

1) Delete invoking task by `exd_tsk`
   The `exd_tsk` terminates the invoking task normally and deletes the task.
   The following describes an example for coding this service call.

   ```c
   #include "kernel.h" /*Standard header file definition*/
   #include "kernel_id.h" /*Header file generated by cfg600px*/
   #pragma task Task1 /*Refer to note*/
   void Task1 (VP_INT exinf); /*Refer to note*/
   
   void Task1 (VP_INT exinf)
   {
     /* ........ */
     exd_tsk (); /*Terminate and delete invoking task*/
   }
   
   #include "kernel.h" /*Standard header file definition*/
   #include "kernel_id.h" /*Header file generated by cfg600px*/
   #pragma task Task1 /*Refer to note*/
   void Task1 (VP_INT exinf); /*Refer to note*/
   void Task1 (VP_INT exinf)
   {
     ID tskid = 8; /*Declares and initializes variable*/
     /* ........ */
     del_tsk ( tskid ); /*Delete other task*/
   }
   
   Note 1 When the invoking task has locked mutexes, the locked state are released at the same time (processing equivalent to `unl_mtx`).
   
   Note 2 These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".

2) Delete another task by `del_tsk`
   The `del_tsk` deletes the task in the DORMANT state indicated by parameter `tskid`.
   This service call can be called from tasks that belong to Trusted Domain.
   The following describes an example for coding this service call.

   ```c
   #include "kernel.h" /*Standard header file definition*/
   #include "kernel_id.h" /*Header file generated by cfg600px*/
   
   void Task1 (VP_INT exinf); /*Refer to note*/
   void Task1 (VP_INT exinf)
   {
     /* ......... */
     del_tsk ( tskid ); /*Delete other task*/
   }
   
   Note These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
4.5 Activate Task

The RI600PX provides two types of interfaces for task activation: queuing an activation request queuing and not queuing an activation request.

4.5.1 Activate task with queuing

A task (queuing an activation request) is activated by issuing the following service call from the processing program.

- `act_tsk`, `iact_tsk`

These service calls move the task specified by parameter `tskid` from the DORMANT state to the READY state. As a result, the target task is queued at the end on the ready queue corresponding to the initial priority and becomes subject to scheduling by the RI600PX.

If the target task has been moved to a state other than the DORMANT state when this service call is issued, this service call does not move the state but increments the activation request counter (by added 1 to the activation request counter).

The following describes an example for coding these service calls.

```c
#include    "kernel.h"          /*Standard header file definition*/
#include    "kernel_id.h"       /*Header file generated by cfg600px*/
#pragma task Task1
/*Refer to note 3*/
void Task1 (VP_INT exinf);          /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
  ID tskid = 8;          /*Declares and initializes variable*/
  /* ......... */
  act_tsk (tskid);            /*Activate task (queues an activation request)*/
  /* ......... */
}
```

Note 1 The activation request counter managed by the RI600PX is configured in 8-bit widths. If the number of activation requests exceeds the maximum count value 255 as a result of issuing this service call, the counter manipulation processing is therefore not performed but “E_QOVR” is returned.

Note 2 Extended information specified at creating the task is passed to the task activated by issuing these service calls.

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
4.5.2 Activate task without queuing

A task (not queuing an activation request) is activated by issuing the following service call from the processing program.

- `sta_tsk, ista_tsk`

These service calls move the task specified by parameter `tskid` from the DORMANT state to the READY state. As a result, the target task is queued at the end on the ready queue corresponding to the initial priority and becomes subject to scheduling by the RI600PX.

This service call does not perform queuing of activation requests. If the target task is in a state other than the DORMANT state, the status manipulation processing for the target task is therefore not performed but “E_OBJ” is returned.

Specify for parameter `stacd` the extended information transferred to the target task.

The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task

void Task1 (VP_INT exinf) { /*Refer to note*/
    ID tskid = 8; /*Declares and initializes variable*/
    VP_INT stacd = 123; /*Declares and initializes variable*/
    /* ........ */
    sta_tsk (tskid, stacd); /*Activate task (does not queue an activation */
                             /*request)*/
    /* ........ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
```

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
```

#pragma task

```c
void Task1 (VP_INT exinf) { /*Refer to note*/
    ID tskid = 8; /*Declares and initializes variable*/
    VP_INT stacd = 123; /*Declares and initializes variable*/
    /* ........ */
    sta_tsk (tskid, stacd); /*Activate task (does not queue an activation */
                             /*request)*/
    /* ........ */
}
```
4.6 Cancel Task Activation Requests

An activation request is cancelled by issuing the following service call from the processing program.

- **can_act, ican_act**

  This service call cancels all of the activation requests queued to the task specified by parameter tskid (sets the activation request counter to 0).

  When this service call is terminated normally, the number of cancelled activation requests is returned.

  The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1               /*Refer to note 2*/
void Task1 (VP_INT exinf);      /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ER_UINT ercd;                   /*Declares variable*/
    ID tskid = 8;                   /*Declares and initializes variable*/
    /* ............ */
    ercd = can_act (tskid);         /*Cancel task activation requests*/
    if (ercd >= 0) {
        /* ............ */           /*Normal termination processing*/
    }
    /* ............ */
}
```

**Note 1**  This service call does not perform status manipulation processing but performs the setting of activation request counter. Therefore, the task does not move from a state such as the READY state to the DORMANT state.

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
4.7 Terminate Task

4.7.1 Terminate invoking task

The invoking task is terminated by issuing the following service call from the processing program.

- **ext_tsk**
  This service call moves the invoking task from the RUNNING state to the DORMANT state. As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. If an activation request has been queued to the invoking task (the activation request counter > 0) when this service call is issued, this service call moves the task from the RUNNING state to the DORMANT state, decrements the wake-up request counter (by subtracting 1 from the activation request counter), and then moves the task from the DORMANT state to the READY state. The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1
   void Task1 (VP_INT exinf) {      /*Refer to note 3*/
      /* ......... */
      ext_tsk ();                     /*Terminate invoking task*/
   }
```

**Note 1** When the invoking task has locked mutexes, the locked state are released at the same time (processing equivalent to `unl_mbx`).

**Note 2** When the return instruction is issued in a task, the same processing as `ext_tsk` is performed.

**Note 3** These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.
4.7.2 Terminate Another task

- ter_tsk

This service call forcibly moves the task specified by parameter tskid to the DORMANT state. As a result, the target task is excluded from the RI600PX scheduling subject.

If an activation request has been queued to the target task (the activation request counter > 0) when this service call is issued, this service call moves the task to the DORMANT state, decrements the activation request counter (by subtracting 1 from the wake-up request counter), and then moves the task from the DORMANT state to the READY state.

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8;              /*Declares and initializes variable*/
    /* .......... */
    ter_tsk (tskid);                /*Terminate task*/
    /* .......... */
}
```

Note 1 When the target task has locked mutexes, the locked state are released at the same time (processing equivalent to unl_mtx).

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
4.8 Change Task Priority

The priority is changed by issuing the following service call from the processing program.

- chg_pri, ichg_pri
  This service call changes the base priority of the task specified by parameter tskid to a value specified by parameter tskpri.
  The changed base priority is effective until the task terminates or this service call is issued. When next the task is activated, the base priority is the initial priority which is specified at the task creation.
  This service call also changes the current priority of the target task to a value specified by parameter tskpri. However, the current priority is not changed when the target task has locked mutexes.
  If the target task has locked mutexes or is waiting for mutex to be locked and if tskpri is higher than the ceiling priority of either of the mutexes, this service call returns “E_ILUSE”.
  When the current priority is changed, the following state variations are generated.

1) When the target task is in the RUNNING or READY state.
   This service call re-queues the task at the end of the ready queue corresponding to the priority specified by parameter tskpri.

2) When the target task is queued to a wait queue of the object with TA_TPRI or TA_CEILING attribute.
   This service call re-queues the task to the wait queue corresponding to the priority specified by parameter tskpri.
   When two or more tasks of same current priority as this service call re-queues the target task at the end among their tasks.

Example
When three tasks (task A: priority level 10, task B: priority level 11, task C: priority level 12) are queued to the semaphore wait queue in the order of priority, and the priority level of task B is changed from 11 to 9, the wait order will be changed as follows.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
  ID tskid = 8;              /*Declares and initializes variable*/
  PRI tskpri = 9;             /*Declares and initializes variable*/

  /* ............ */
  chg_pri (tskid, tskpri);        /*Change task priority*/

  /* ............ */
}
```

Note 1 For current priority and base priority, refer to “8.2.2 Current priority and base priority”.

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
4.9 Reference Task Priority

A task priority is referenced by issuing the following service call from the processing program.

- `get_pri, iget_pri`
  Stores current priority of the task specified by parameter `tskid` in the area specified by parameter `p_tskpri`.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8;              /*Declares and initializes variable*/
    PRI p_tskpri;               /*Declares variable*/

    /* .......... */
    get_pri (tskid, &p_tskpri); /*Reference task priority*/
    /* .......... */
}
```

**Note 1**  For current priority and base priority, refer to "8.2.2 Current priority and base priority".

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
4.10 Reference Task State

4.10.1 Reference task state

A task status is referenced by issuing the following service call from the processing program.

- ref_tsk, iref_tsk
  Stores task state packet (current state, current priority, etc.) of the task specified by parameter tskid in the area specified by parameter pk_rtsk.
  The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8;              /*Declares and initializes variable*/
    T_RTSK pk_rtsk;                /*Declares data structure*/
    STAT tskstat;                /*Declares variable*/
    PRI tskpri;                  /*Declares variable*/
    PRI tskbpri;                 /*Declares variable*/
    STAT tskwait;                /*Declares variable*/
    ID wobjid;                   /*Declares variable*/
    TMO lefttmo;                 /*Declares variable*/
    UINT acctcnt;                /*Declares variable*/
    UINT wupcnt;                 /*Declares variable*/
    UINT suscnt;                 /*Declares variable*/

    /* ........ */

    ref_tsk (tskid, &pk_rtsk);  /*Reference task state*/

    tskstat = pk_rtsk.tskstat;  /*Reference current state*/
    tskpri = pk_rtsk.tskpri;    /*Reference current priority*/
    tskbpri = pk_rtsk.tskbpri;  /*Reference base priority*/
    tskwait = pk_rtsk.tskwait;  /*Reference reason for waiting*/
    wobjid = pk_rtsk.wobjid;    /*Reference object ID number for which the */
    lefttmo = pk_rtsk.lefttmo;  /*Reference remaining time until time-out*/
    acctcnt = pk_rtsk.acctcnt;  /*Reference activation request count*/
    wupcnt = pk_rtsk.wupcnt;    /*Reference wake-up request count*/
    suscnt = pk_rtsk.suscnt;    /*Reference suspension count*/

    /* ........ */
}
```

Note 1 For details about the task state packet, refer to "[Task state packet: T_RTSK]".
Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
### 4.10.2 Reference task state (simplified version)

A task status (simplified version) is referenced by issuing the following service call from the processing program.

- ref_tst, iref_tst

Stores task state packet (current state, reason for waiting) of the task specified by parameter tskid in the area specified by parameter pk_rtst.

Used for referencing only the current state and reason for wait among task information.

Response becomes faster than using ref_tsk or iref_tsk because only a few information items are acquired.

The following describes an example for coding these service calls.

```c
#include "kernel.h"   /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf); /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8;              /*Declares and initializes variable*/
    T_RTST pk_rtst;            /*Declares data structure*/
    STAT tskstat;              /*Declares variable*/
    STAT tskwait;              /*Declares variable*/

    /* ......... */
    ref_tst (tskid, &pk_rtst); /*Reference task state (simplified version)*/

    tskstat = pk_rtst.tskstat; /*Reference current state*/
    tskwait = pk_rtst.tskwait;  /*Reference reason for waiting*/

    /* ......... */
}
```

**Note 1** For details about the task state packet (simplified version), refer to "[Task state packet (simplified version): T_RTST]."

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
CHAPTER 5 TASK DEPENDENT SYNCHRONIZATION FUNCTIONS

This chapter describes the task dependent synchronization functions performed by the RI600PX.

5.1 Outline

The RI600PX provides several task-dependent synchronization functions.

5.2 Put Task to Sleep

5.2.1 Waiting forever

A task is moved to the sleeping state (waiting forever) by issuing the following service call from the processing program.

- slp_tsk

This service call moves the invoking task from the RUNNING state to the WAITING state (sleeping state).

If a wake-up request has been queued to the target task (the wake-up request counter > 0) when this service call is issued, this service call does not move the state but decrements the wake-up request counter (by subtracting 1 from the wake-up request counter).

The sleeping state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sleeping State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A wake-up request was issued as a result of issuing wup_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A wake-up request was issued as a result of issuing iwup_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note*/
void Task1 (VP_INT exinf); /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER ercd; /*Declares variable*/

    /* ......... */
    ercd = slp_tsk (); /*Put task to sleep*/
    if (ercd == E_OK) {
        /* ......... */
    } else if (ercd == E_RLWAI) {
        /* ......... */
    }
    /* ......... */
}
```
5.2.2 With time-out

A task is moved to the sleeping state (with time-out) by issuing the following service call from the processing program.

- `tslp_tsk`

This service call moves the invoking task from the RUNNING state to the WAITING state with time-out(sleeping state).

As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject.

If a wake-up request has been queued to the target task (the wake-up request counter > 0) when this service call is issued, this service call does not move the state but decrements the wake-up request counter (by subtracting 1 from the wake-up request counter).

The sleeping state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sleeping State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A wake-up request was issued as a result of issuing <code>wup_tsk</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A wake-up request was issued as a result of issuing <code>iwup_tsk</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include "kernel.h"          /*Standard header file definition*/
#include "kernel_id.h"        /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);   /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;               /*Declares variable*/
    TMO tmout = 3600;       /*Declares and initializes variable*/
    /* ......... */
    ercd = tslp_tsk (tmout);    /*Put task to sleep*/
    if (ercd == E_OK) {
        /* ............ */
    } else if (ercd == E_RLWAI) {
        /* ............ */
    } else if (ercd == E_TMOUT) { /*Time-out processing*/
        /* ............ */
    } else { /* ............ */
    }

    /* ............ */
}
```

Note 1 When `TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `slp_tsk` will be executed.

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
5.3 Wake-up Task

A task is woken up by issuing the following service call from the processing program.

- `wup_tsk`, `iwup_tsk`
  These service calls cancel the WAITING state (sleeping state) of the task specified by parameter `tskid`. As a result, the target task is moved from the sleeping state to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  If the target task is in a state other than the sleeping state when this service call is issued, this service call does not move the state but increments the wake-up request counter (by added 1 to the wake-up request counter).
  The following describes an example for coding these service calls.

```
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1               /*Refer to note 2*/
void Task1 (VP_INT exinf);      /*Refer to note 2*/
void Task1 (VP_INT exinf) {
    ID tskid = 8;              /*Declares and initializes variable*/
    /* ........... */
    wup_tsk (tskid);                /*Wake-up task*/
    /* ........... */
}
```

Note 1  The wake-up request counter managed by the RI600PX is configured in 8-bit widths. If the number of wake-up requests exceeds the maximum count value 255 as a result of issuing this service call, the counter manipulation processing is therefore not performed but "E_QOVR" is returned.

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
5.4 Cancel Task Wake-up Requests

A wake-up request is cancelled by issuing the following service call from the processing program.

- **can_wup, ican_wup**

These service calls cancel all of the wake-up requests queued to the task specified by parameter *tskid* (the wake-up request counter is set to 0).

When this service call is terminated normally, the number of cancelled wake-up requests is returned. The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf) {
    ER_UINT ercd;                   /*Declares variable*/
    ID tskid = 8;              /*Declares and initializes variable*/
    /* .......... */
    ercd = can_wup (tskid);         /*Cancel task wake-up requests*/
    if (ercd >= 0) {
        /* .......... */             /*Normal termination processing*/
    }
    /* .......... */
}
```

**Note** These statements are unnecessary for the task which is created by the system configuration file because the *cfg600px* generates these statement into the "kernel_id.h".
5.5 Forcibly Release Task from Waiting

The WAITING state is forcibly cancelled by issuing the following service call from the processing program.

- rel_wai, irel_wai
  
  These service calls forcibly cancel the WAITING state of the task specified by parameter tskid.
  As a result, the target task unlinked from the wait queue and is moved from the WAITING state to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  “E_RLWAI” is returned from the service call that triggered the move to the WAITING state (slp_tsk, wai_sem, or the like) to the task whose WAITING state is cancelled by this service call.
  
  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                 /*Refer to note 3*/

void Task1 (VP_INT exinf);          /*Refer to note 3*/
void Task1 (VP_INT exinf)
{
  ID tskid = 8;              /*Declares and initializes variable*/
  /* .......... */
  rel_wai (tskid);                /*Release task from waiting*/
  /* .......... */
}
```

Note 1  This service call does not perform queuing of forced cancellation requests. If the target task is in a state other than the WAITING or WAITING-SUSPENDED state, “E_OBJ” is returned.

Note 2  The SUSPENDED state is not cancelled by these service calls.

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

5.6 Suspend Task

A task is moved to the SUSPENDED state by issuing the following service call from the processing program.

- sus_tsk, isus_tsk

These service calls move the target task specified by parameter tskid from the RUNNING state to the SUSPENDED state, from the READY state to the SUSPENDED state, or from the WAITING state to the WAITING-SUSPENDED state.

If the target task has moved to the SUSPENDED or WAITING-SUSPENDED state when this service call is issued, these service calls return E_QOVR.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8;              /*Declares and initializes variable*/
    /* ......... */
    sus_tsk (tskid);                /*Suspend task*/
    /* ......... */
}
```

Note: These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
5.7 Resume Suspended Task

5.7.1 Resume suspended task

The SUSPENDED state is cancelled by issuing the following service call from the processing program.

- **rsm_tsk, irsm_tsk**
  These service calls move the target task specified by parameter `tskid` from the SUSPENDED state to the READY state, or from the WAITING-SUSPENDED state to the WAITING state.
  The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 3*/
void Task1 (VP_INT exinf); /*Refer to note 3*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8; /*Declares and initializes variable*/
    /* ........ */
    rsm_tsk (tskid); /*Resume suspended task*/
    /* ........ */
}
```

**Note 1** This service call does not perform queuing of cancellation requests. If the target task is in a state other than the SUSPENDED or WAITING-SUSPENDED state, “E_OBJ” is therefore returned.

**Note 2** The RI600PX does not support queing of suspend request. The behavior of the `frsm_tsk` and `ifrsm_tsk`, that can release from the SUSPENDED state even if suspend request has been queued, are same as `rsm_tsk` and `irsm_tsk`.

**Note 3** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”. 
5.7.2 Forcibly resume suspended task

The SUSPENDED state is forcibly cancelled by issuing the following service calls from the processing program.

- frsm_tsk, ifrsm_tsk
  These service calls move the target task specified by parameter tskid from the SUSPENDED state to the READY state, or from the WAITING-SUSPENDED state to the WAITING state.
  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/

void Task1 (VP_INT exinf);          /*Refer to note 3*/

void Task1 (VP_INT exinf)           /*Refer to note 3*/
{
    ID tskid = 8;              /*Declares and initializes variable*/

    /* ............ */
    frsm_tsk (tskid);                /*Forcibly resume suspended task*/

    /* ............ */
}
```

Note 1 This service call does not perform queuing of cancellation requests. If the target task is in a state other than the SUSPENDED or WAITING-SUSPENDED state, "E_OBJ" is returned.

Note 2 The RI600PX does not support queing of suspend request. Therefore, the behavior of these service calls are same as rsm_tsk and irsm_tsk.

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

5.8 Delay Task

A task is moved to the delayed state by issuing the following service call from the processing program.

- **dly_tsk**
  This service call moves the invoking task from the RUNNING state to the WAITING state (delayed state). As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. The delayed state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Delayed State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay time specified by parameter <em>dlytim</em> has elapsed.</td>
<td><strong>E_OK</strong></td>
</tr>
<tr>
<td>Forced release from waiting (accept <em>rel_wai</em> while waiting).</td>
<td><strong>E_RLWAI</strong></td>
</tr>
<tr>
<td>Forced release from waiting (accept <em>irel_wai</em> while waiting).</td>
<td><strong>E_RLWAI</strong></td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note 2*/

void Task1 (VP_INT exinf) {          /*Refer to note 2*/

  ER ercd;                   /*Declares variable*/
  RELTIM dlytim = 3600;          /*Declares and initializes variable*/
  /* ............ */

  ercd = dly_tsk (dlytim);        /*Delay task*/
  if (ercd == E_OK) {
    /* ............ */
  } else if (ercd == E_RLWAI) {
    /* ............ */
  }
  /* ............ */
}
```

**Note 1** When 0 is specified as *dlytim*, the delay time is up to next base clock interrupt generation.

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
5.9 Differences Between Sleep with Time-out and Delay

There are differences between “Sleep with time-out (5.2.2 With time-out)” and “Delay (5.8 Delay Task)” as shown in Table 5-1.

Table 5-1 Differences Between “Sleep with time-out” and “Delay”

<table>
<thead>
<tr>
<th>Service call that causes status change</th>
<th>Sleep with time-out</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>tslp_tsk</td>
<td>tsk</td>
<td>dly_tsk</td>
</tr>
<tr>
<td>Return value when time has elapsed</td>
<td>E_TMOOUT</td>
<td>E_OK</td>
</tr>
<tr>
<td>Operation when wup_tsk or iwup_tsk is issued</td>
<td>Wake-up</td>
<td>Queues the wake-up request (time elapse wait is not cancelled).</td>
</tr>
</tbody>
</table>
CHAPTER 6 TASK EXCEPTION HANDLING FUNCTIONS

This chapter describes the task exception handling functions performed by the RI600PX.

6.1 Outline

When task exception is requested to a task, the task exception handling routine defined for the task is invoked. The requested exception code is passed to the task exception handling routine. By this function, exception handling in a task can be implemented easily. The following shows the service calls as the task exception handling functions.

Table 6-1 Task Exception Handling Functions

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>def_tex</td>
<td>Define task exception handling</td>
<td>Task</td>
</tr>
<tr>
<td></td>
<td>routine</td>
<td></td>
</tr>
<tr>
<td>ras_tex</td>
<td>Raise task exception</td>
<td>Task</td>
</tr>
<tr>
<td>iras_tex</td>
<td>Raise task exception</td>
<td>Non-task</td>
</tr>
<tr>
<td>dis_tex</td>
<td>Disable task exception</td>
<td>Task</td>
</tr>
<tr>
<td>ena_tex</td>
<td>Enable task exception</td>
<td>Task</td>
</tr>
<tr>
<td>sns_tex</td>
<td>Reference task exception disabled</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>ref_tex</td>
<td>Reference task exception state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_tex</td>
<td>Reference task exception state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
6.2 Task Exception Handling Routines

Exception handling according to the requested exception code should be implemented in a task exception handling routine.

6.2.1 Basic form of task exception handling routines

The following shows the basic form of ask exception handling routines.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma taskexception Texrtn1       /*Refer to note 3*/
void Texrtn1 ( TEXPTN texptn, VP_INT exinf );   /*Refer to note 3*/
void Texrtn1 ( TEXPTN texptn, VP_INT exinf )
{
    /* ........ */
}
```

Note 1 The accepted exception code is passed to `texptn`.

Note 2 The extend information defined at creating the task is passed to `exinf`.

Note 3 These statements are unnecessary for the task exception handling routine which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
### 6.2.2 Internal processing of task exception handling routine

- **Stack**
  Task exception handling routines use user stacks for the task.

- **Service call**
  Tasks can issue service calls whose "Useful range" is "task".

- **PSW register when processing is started**

  Table 6-2 PSW Register When Task Exception Handling Routine Processing is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td></td>
<td>Same as IPL in the task just before the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>task exception handling routine starts.</td>
</tr>
<tr>
<td>PM</td>
<td>1</td>
<td>User mode</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>User stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **FPSW register when processing is started**
  When setting of **Task context register (context)** in **System Information (system)** includes "FPSW", the FPSW when processing is started is shown in **Table 6-3**. The FPSW when processing is undefined in other cases.

  Table 6-3 FPSW Register When Task Processing is Started

<table>
<thead>
<tr>
<th>Compiler options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-round nearest</td>
<td>0x00000100 (Only DN bit is 1.)</td>
</tr>
<tr>
<td>-denormalize off</td>
<td>0x00000101 (Only DN bit and RM bit are 1.)</td>
</tr>
<tr>
<td></td>
<td>on 0</td>
</tr>
<tr>
<td></td>
<td>on 1 (Only RM bit is 1.)</td>
</tr>
</tbody>
</table>
6.2.3 The starting conditions of task exception handling routines

When all the following conditions are fulfilled about a task which is scheduled according to “16.4 Task Scheduling Method”, the RI600PX starts the task exception handling routine in stead of the task itself. When the task exception handling routine is finished, execution of the task itself is resumed.

Table 6-4 The Starting Conditions of Task Exception Handling Routines

<table>
<thead>
<tr>
<th>No.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The task is in the task exception enabled state.</td>
</tr>
<tr>
<td>2</td>
<td>The pending exception code of the task is not 0.</td>
</tr>
</tbody>
</table>

The exception code is represented by TEXPTN type bit pattern. When task exception is requested by ras_tex or iras_tex, the pending exception code for the corresponded task is renewed to the logical add with specified exception code.

Tasks are in one of the state of task exception disabled state or task exception enabled state. Immediately after starting tasks, the task is in task exception disabled state. And the task is in task exception disabled state while the task exception handling routine is not defined. When ena_tex is called, the invoking task enters in task exception enabled state. When dis_tex is called, the invoking task enters in task exception disabled state.

When the task exception handling routine is started, the task enters in task exception disabled state. When the task exception handling routine is finished, the task enters in task exception enabled state.

Table 6-5 Operations to Disable Task Exception

<table>
<thead>
<tr>
<th>No.</th>
<th>Operating</th>
<th>Target task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Activate Task</td>
<td>Activated task</td>
</tr>
<tr>
<td>2</td>
<td>dis_tex</td>
<td>Invoking task</td>
</tr>
<tr>
<td>3</td>
<td>Starting of task exception handling routine</td>
<td>Concerned task</td>
</tr>
<tr>
<td>4</td>
<td>Cancel a Definition of Task Exception Handling Routine</td>
<td>Concerned task</td>
</tr>
</tbody>
</table>

Table 6-6 Operations to Enable Task Exception

<table>
<thead>
<tr>
<th>No.</th>
<th>Operating</th>
<th>Target task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ena_tex</td>
<td>Invoking task</td>
</tr>
<tr>
<td>2</td>
<td>Finishing of task exception handling routine</td>
<td>Concerned task</td>
</tr>
</tbody>
</table>
6.3 Define Task Exception Handling Routine

Task exception handling routines are defined by one of the following methods.

1) Definition by the system configuration file
   The task exception handling routine can be defined by the static API "task[]", which creates a task, described in the
   system configuration file.
   Refer to "20.10 Task Information (task[])" for the details of "task[]".

2) Definition by def_tex
   The def_tex defines a task exception handling routine for the task indicated by parameter tskid according to the
   content of parameter pk_dtex.
   The information specified is shown below.
   - Task exception handling routine attribute (texatr)
     Only TA_HLNG can be specified as texatr.
   - Task exception handling routine start address (texrtn)

   This service call can be called from tasks that belong to Trusted Domain.
   The following describes an example for coding def_tex.

```c
#include "kernel.h"      /*Standard header file definition*/
#include "kernel_id.h"    /*Header file generated by cfg600px*/
extern void Texrtn1 ( TEXPTN texptn, VP_INT exinf );

#pragma task Task1       /*Refer to note*/
void Task1 (VP_INT exinf);  /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER ercd;                  /*Declares variable*/
    ID tskid = TSK_SELF;      /*Declares and initializes variable*/
    T_DTEX pk_dtex = {
        TA_HLNG,            /*Task exception handling routine attribute (texatr)*//*Declares and initializes variable*/
        (FP)Texrtn1         /*Start address (texrtn)*/
    };
    /* ......... */
    ercd = def_tex ( tskid, &pk_dtex ); /*Define task exception handling routine*/
    /* ......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the
cfg600px generates these statement into the "kernel_id.h".
6.4 Cancel a Definition of Task Exception Handling Routine

When NULL is specified as parameter pk_dtex in the def_tex, the def_tex cancels a definition of the task exception handling routine for the task indicated by parameter tskid.

The def_tex can be called from tasks that belong to Trusted Domain.

The following describes an example for coding to cancel the a definition of the task exception handling routine by using def_tex.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);         /*Refer to note*/
void Task1 (VP_INT exinf)           
{
    ER ercd;                        /*Declares variable*/
    ID tskid = TSK_SELF;            /*Declares and initializes variable*/
    /* .......... */
    ercd = def_tex ( tsid, (T_DTEX *)NULL ); /*Cancel a definition*/
    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

6.5 Request Task Exception

- ras_tex, iras_tex
These service calls requests task exception handling for the task indicated by parameter tskid. The task pending exception code for the task is ORed with the value indicated by parameter rasptn. When all the conditions described in “6.2.3 The starting conditions of task exception handling routines” are fulfilled by this service call, the RI600PX starts the task exception handling routine.

The following describes an example for coding this service call.

```
#include    "kernel.h"          /*Standard header file definition*/
#include    "kernel_id.h"       /*Header file generated by cfg600px*/
#pragma task Task1             /*Refer to note*/
void Task1 (VP_INT exinf);     /*Refer to note*/
void Task1 (VP_INT exinf) {
    ID tskid = 8;              /*Declares and initializes variable*/
    TEXPTN rasptn = 0x00000001UL;  /*Declares and initializes variable*/
    /* ............... */
    ras_tex ( tskid, rasptn );  /*Request task exception*/
    /* ............... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

6.6 Disable and Enable Task Exception

The `dis_tex` disables task exception for the invoking task, and the `ena_tex` enables task exception for the invoking task. The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/
void TaskI (VP_INT exinf);         /*Refer to note*/
void TaskI (VP_INT exinf)
{
    /* ......... */
    ena_tex ();                     /*Disable task exception*/
    /* ......... */                 /*Task exception enabled state*/
    dis_tex ();                     /*Enable task exception*/
    /* ......... */
}
```

Note: These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statements into the "kernel_id.h".
6.7 Reference Task Exception Disabled State

It may be necessary to refer to task exception disabled state for the task of a calling agency in functions that are called from two or more tasks and handlers. In this case, sns_tex is useful.

- sns_tex

This service call returns TRUE when the task in the RUNNING state is in the task exception disabled state, and when other, this service call returns FALSE.

The following describes an example for coding this service call.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/

void CommonFunc ( void );

void CommonFunc ( void )
{
    BOOL ctx;       /*Declares variable*/
    BOOL tex;       /*Declares variable*/

    /* ........... */

    ctx = sns_ctx ( );              /*Reference context type*/

    if ( ctx == TRUE ) {
        /* ........... */             /*Processing for non-task context*/
    } else if ( ctx == FALSE ) {
        /* ........... */             /*Processing for task context*/
        tex = sns_tex ();           /*Reference task exception disabled state*/

        if ( tex == TRUE ) {
            /* ........... */         /*The invoking task is in the task*/
        } else if ( tex == FALSE ) {
            /* ........... */         /*The invoking task is in the task*/
        }
    } /* ........... */
} /* ........... */
```
6.8 Reference Task Exception State

- `ref_tex`, `iref_tex`
  Stores task exception state packet of the task specified by parameter `tskid` in the area specified by parameter `pk_rtex`. The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf); /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID tskid = 8; /*Declares and initializes variable*/
    T_RTEX pk_rtex; /*Declares data structure*/
    STAT texstat; /*Declares variable*/
    TEXPTN texptn; /*Declares variable*/

    /* ......... */
    ref_tex (tskid, &pk_rtex); /*Reference task exception state*/

    texstat = pk_rtex.texstat; /*Reference task exception handling state*/
    texptn = pk_rtex.texptn; /*Reference pending exception code*/

    /* ......... */
}
```

Note 1  For details about the task exception state packet, refer to "[Task exception state packet: T_RTEX]."

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
This chapter describes the synchronization and communication functions performed by the RI600PX.

### 7.1 Outline

The synchronization and communication functions of the RI600PX consist of Semaphores, Eventflags, Data Queues, and Mailboxes that are provided as means for realizing exclusive control, queuing, and communication among tasks.

### 7.2 Semaphores

In the RI600PX, non-negative number counting semaphores are provided as a means (exclusive control function) for preventing contention for limited resources (hardware devices, library function, etc.) arising from the required conditions of simultaneously running tasks. The following shows a processing flow when using a semaphore.

Figure 7-1 Processing Flow (Semaphore)

```
Task

↑       Acquire semaphore resource

Exclusive control period

↓       Release semaphore resource
```
7.2.1 Create semaphore

Semaphores are created by one of the following methods.

1) Creation by the system configuration file
   The static API "semaphore[]" described in the system configuration file creates a semaphore.
   Refer to "20.11 Semaphore Information (semaphore[])" for the details of "semaphore[]."

2) Creation by cre_sem or acre_sem
   The cre_sem creates a semaphore with semaphore ID indicated by parameter semid according to the content of parameter pk_csem.
   The acre_sem creates a semaphore according to the content of parameter pk_csem, and returns the created semaphore ID.
   The information specified is shown below.
   - Semaphore attribute (sematr)
     The following informations are specified as sematr:
     - The order of task wait queue (FIFO order or task current priority order)
   - Initial semaphore count (isemcnt)
   - Maximum semaphore count (maxsem)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_sem as a representative.

```c
#include    "kernel.h"             /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/
void Task1 (VP_INT exinf);         /*Refer to note*/
void Task1 (VP_INT exinf) {
  ER            semid;             /*Declares variable*/
  T_CSEM        pk_csem = {         /*Declares and initializes variable*/
     TA_TFIFO,                  /*Semaphore attribute (sematr)*/
     1,                        /*Initial semaphore count (isemcnt)*/
     0                         /*Maximum semaphore count (maxsem)*/
  };
  /* ............ */
  semid = acre_sem ( &pk_csem );   /*Create semaphore/
  /* ............ */
}
```

Note  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.2.2 Delete semaphore

- del_sem
  This service call deletes the semaphore specified by parameter semid.
  When there are waiting tasks for the target semaphore by using wai_sem or twai_sem, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the wai_sem or twai_sem.
  This service call can be called from tasks that belong to Trusted Domain.
  The following describes an example for coding this service call.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task

void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ID semid = 8;          /*Declares and initializes variable*/

    /* ........... */

    ercd = del_sem ( semid );   /*Delete semaphore*/

    /* ........... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.2.3 Acquire semaphore resource

A resource is acquired (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- **wai_sem** (Wait)
- **pol_sem, ipol_sem** (Polling)
- **twai_sem** (Wait with time-out)

- **wai_sem** (Wait)
  This service call acquires a resource from the semaphore specified by parameter `semid` (subtracts 1 from the semaphore counter).

  When no resources are acquired from the target semaphore when this service call is issued (no available resources exist), this service call does not acquire resources but queues the invoking task to the target semaphore wait queue and moves it from the **RUNNING** state to the **WAITING** state (resource acquisition wait state).
  The **WAITING** state for a semaphore resource is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Semaphore Resource Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>sig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>isig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_sem</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);          /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
  ER ercd;               /*Declares variable*/
  ID semid = 1;          /*Declares and initializes variable*/
  /* .......... */
  ercd = wai_sem (semid);    /*Acquire semaphore resource*/
  if (ercd == E_OK) {
    /* .......... */    /*Normal termination processing*/
    sig_sem (semid);    /*Release semaphore resource*/
  } else if (ercd == E_RLWAI) {
    /* .......... */    /*Forced termination processing*/
  }
  /* .......... */
}
```

**Note 1** Invoking tasks are queued to the target semaphore wait queue in the order defined at creating the semaphore (FIFO order or current priority order).

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
- pol_sem, ipol_sem (Polling)

These service calls acquire a resource from the semaphore specified by parameter `semid` (subtracts 1 from the semaphore counter).

If a resource could not be acquired from the target semaphore (semaphore counter is set to 0) when these service calls are issued, the counter manipulation processing is not performed but “E_TMOUT” is returned.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1      /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER ercd;                   /*Declares variable*/
    ID semid = 1;              /*Declares and initializes variable*/
    /* ......... */
    ercd = pol_sem (semid);         /*Acquire semaphore resource*/
    if (ercd == E_OK) {
        /* ......... */            /*Polling success processing*/
        sig_sem ( semid );        /*Release semaphore resource*/
    } else if (ercd == E_TMOUT) {
        /* ......... */            /*Polling failure processing*/
    }
    /* ......... */
}
```

Note: These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- `twai_sem` (Wait with time-out)

This service call acquires a resource from the semaphore specified by parameter `semid` (subtracts 1 from the semaphore counter).

If no resources are acquired from the target semaphore when service call is issued this (no available resources exist), this service call does not acquire resources but queues the invoking task to the target semaphore wait queue and moves it from the RUNNING state to the WAITING state with time-out (resource acquisition wait state).

The WAITING state for a semaphore resource is cancelled in the following cases:

<table>
<thead>
<tr>
<th>WAITING State for a Semaphore Resource Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>sig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>isig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include "kernel.h" // Standard header file definition
#include "kernel_id.h" // Header file generated by cfg600px
#pragma task Task1 // Refer to note 3*
void Task1 (VP_INT einfo); // Refer to note 3*

void Task1 (VP_INT einfo)
{
    ER ercd; // Declares variable*/
    ID semid = 1; // Declares and initializes variable*/
    TMO tmout = 3600; // Declares and initializes variable*/

    /* ........ ....... */

    ercd = twai_sem (semid, tmout); /* Acquire semaphore resource*/
    if (ercd == E_OK) {
    /* ........ ....... */ /* Normal termination processing*/
        sig_sem (semid); /* Release semaphore resource*/
    } else if (ercd == E_RLWAI) {
    /* ........ ....... */ /* Forced termination processing*/
        } else if (ercd == E_TMOUT) {
        /* ........ ....... */ /* Time-out processing*/
    }

    /* ........ ....... */
}
```

Note 1  Invoking tasks are queued to the target semaphore wait queue in the order defined at creating the semaphore (FIFO order or current priority order).

Note 2  `TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `wai_sem` will be executed. When `TMO_POL` is specified, processing equivalent to `pol_sem` will be executed.

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.
### 7.2.4 Release semaphore resource

A resource is released by issuing the following service call from the processing program.

```c
- sig_sem, isig_sem
```

These service calls releases the resource to the semaphore specified by parameter `semid` (adds 1 to the semaphore counter).

If a task is queued in the wait queue of the target semaphore when this service call is issued, the counter manipulation processing is not performed but the resource is passed to the relevant task (first task of wait queue).

As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (WAITING state for a semaphore resource) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

The following describes an example for coding these service calls.

```c
#include "kernel.h"          /* Standard header file definition */
#include "kernel_id.h"       /* Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf);     /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;               /* Declares variable */
    ID semid = 1;          /* Declares and initializes variable */

    ercd = wai_sem (semid);    /* Acquire semaphore resource */

    if (ercd == E_OK) {
        /* .......... */       /* Normal termination processing */

            sig_sem (semid);  /* Release semaphore resource */
    } else if (ercd == E_RLWAI) {
        /* .......... */       /* Forced termination processing */
    }

    /* .......... */
}
```

**Note 1** With the RI600PX, the maximum possible number of semaphore resources (maximum resource count) is defined during configuration. If the number of resources exceeds the specified maximum resource count, this service call therefore does not release the acquired resources (addition to the semaphore counter value) but returns E_QOVR.

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.2.5 Reference semaphore state

A semaphore status is referenced by issuing the following service call from the processing program.

- ref_sem, iref_sem
  Stores semaphore state packet (ID number of the task at the head of the wait queue, current resource count, etc.) of the semaphore specified by parameter semid in the area specified by parameter pk_rsem.
  The following describes an example for coding these service calls.

```c
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf); /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID semid = 1;            /*Declares and initializes variable*/
    T_RSEM pk_rsem;          /*Declares variable*/
    ID wtskid;               /*Declares variable*/
    UINT semcnt;             /*Declares variable*/

    /* ......... */

    ref_sem (semid, &pk_rsem); /*Reference semaphore state*/

    wtskid = pk_rsem.wtskid;  /*Reference ID number of the task at the */
                              /*head of the wait queue*/
    semcnt = pk_rsem.semcnt;  /*Reference current resource count*/

    /* ......... */
}
```

Note 1 For details about the semaphore state packet, refer to "[Semaphore state packet: T_RSEM]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.3 Eventflags

The RI600PX provides 32-bit eventflags as a queuing function for tasks, such as keeping the tasks waiting for execution, until the results of the execution of a given processing program are output. The following shows a processing flow when using an eventflag.

![Processing Flow (Eventflag)](image-url)
7.3.1 Create eventflag

Eventflags are created by one of the following methods.

1) Creation by the system configuration file
   The static API “flag[]” described in the system configuration file creates a eventflag.
   Refer to “20.12 Eventflag Information (flag[])” for the details of “flag[]”.

2) Creation by cre_flg or acre_flg
   The cre_flg creates a eventflag with eventflag ID indicated by parameter flgid according to the content of parameter pk_cflg.
   The acre_flg creates a eventflag according to the content of parameter pk_cflg, and returns the created eventflag ID.
   The information specified is shown below.
   - Eventflag attribute (flgatr)
     The following informations are specified as flgatr.
     - The order of task wait queue (FIFO order or task current priority order)
     - Whether multiple task can wait on the event flag.
     - Whether the bit pattern of the event flag is cleared to 0 when a task is released from the WAITING state,
   - Initial bit pattern (iflgptn)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_flg as a representative.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1      /*Refer to note*/
void Task1(VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER flgid;              /*Declares variable*/
    T_CFLG pk_cflg = {         /*Declares and initializes variable*/
        TA_TFIFO|TA_WSGL|TA_CLR,    /*Eventflag attribute (flgatr)*/
        0UL                         /*Initial bit pattern (iflgptn)*/
    };                         /*Refer to note*/
    /* ......... */
    flgid = acre_flg ( &pk_cflg ); /*Create eventflag/  
    /* ......... */
}
```

Note: These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
### 7.3.2 Delete Eventflag

- **del_flg**

  This service call deletes the eventflag specified by parameter `flgid`. When there are waiting tasks for the target eventflag by using `wai_flg` or `twai_flg`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `wai_flg` or `twai_flg`.

  This service call can be called from tasks that belong to Trusted Domain.

  The following describes an example for coding this service call.

  ```c
  #include    "kernel.h"              /*Standard header file definition*/
  #include    "kernel_id.h"           /*Header file generated by cfg600px*/
  #pragma task Task1                  /*Refer to note*/
  void Task1 (VP_INT exinf);         /*Refer to note*/
  void Task1 (VP_INT exinf) {
      ID flgid = 8;                       /*Declares and initializes variable*/
      /* .......... */
      ercd = del_flg ( flgid );         /*Delete semaphore*/
      /* .......... */
  }
  
  Note  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.3.3 Set eventflag

A bit pattern is set by issuing the following service call from the processing program.

- `set_flg, iset_flg`

  These service calls set the result of ORing the bit pattern of the eventflag specified by parameter `flgid` and the bit pattern specified by parameter `setptn` as the bit pattern of the target eventflag. After that, these service calls evaluate whether the wait condition of the tasks in the wait queue is satisfied. This evaluation is done in order of the wait queue. If the wait condition is satisfied, the relevant task is unlinked from the wait queue at the same time as bit pattern setting processing. As a result, the relevant task is moved from the WAITING state (WAITING state for an eventflag) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. At this time, the bit pattern of the target event flag is cleared to 0 and this service call finishes processing if the TA_CLR attribute is specified for the target eventflag.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID flgid = 1;              /*Declares and initializes variable*/
    FLGPTN setptn = 0x00000001UL;  /*Declares and initializes variable*/

    /* .......... */
    set_flg (flgid, setptn);        /*Set eventflag*/

    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
### 7.3.4 Clear eventflag

A bit pattern is cleared by issuing the following service call from the processing program.

- **clr_flg, iclr_flg**

  This service call sets the result of ANDing the bit pattern set to the eventflag specified by parameter `flgid` and the bit pattern specified by parameter `clrptn` as the bit pattern of the target eventflag.

  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/
void Task1 (VP_INT exinf);         /*Refer to note*/
void Task1 (VP_INT exinf) {
    ID flgid = 1;              /*Declares and initializes variable*/
    FLGPTN clrptn = 0xFFFFFFFEUL;  /*Declares and initializes variable*/

    /* ............ */

    clr_flg (flgid, clrptn);        /*Clear eventflag*/

    /* ............ */
}
```

**Note** These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.
7.3.5 Check bit pattern

A bit pattern is checked (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- `wai_flg` (Wait)
- `pol_flg`, `ipol_flg` (Polling)
- `twai_flg` (Wait with time-out)

- `wai_flg` (Wait)
  This service call checks whether the bit pattern specified by parameter `waiptn` and the bit pattern that satisfies the required condition specified by parameter `wfmode` are set to the eventflag specified by parameter `flgid`.
  If a bit pattern that satisfies the required condition has been set for the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter `p_flgptn`.
  If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue.
  As a result, the invoking task is unlinked from the ready queue and is moved from the RUNNING state to the WAITING state (WAITING state for an eventflag).
  The WAITING state for an eventflag is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for an Eventflag Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing <code>set_flg</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing <code>iset_flg</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_flg</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

The following shows the specification format of required condition `wfmode`.

- `wfmode = TWF_ANDW`  
  Checks whether all of the bits to which 1 is set by parameter `waiptn` are set as the target eventflag.

- `wfmode = TWF_ORW`  
  Checks which bit, among bits to which 1 is set by parameter `waiptn`, is set as the target eventflag.

The following describes an example for coding this service call.
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 4*/
void Task1 (VP_INT exinf); /*Refer to note 4*/

void Task1 (VP_INT exinf)
{
    ER ercd; /*Declares variable*/
    ID flgid = 1; /*Declares and initializes variable*/
    FLGPTN waiptn = 14; /*Declares and initializes variable*/
    MODE wfmode = TWF_ANDW; /*Declares and initializes variable*/
    FLGPTN p_flgptn; /*Declares variable*/

    /* ............ */
    /*Wait for eventflag*/
    ercd = wai_flg (flgid, waiptn, wfmode, &p_flgptn);

    if (ercd == E_OK) {
        /* ............ */ /*Normal termination processing*/
    } else if (ercd == E_RLWAI) {
        /* ............ */ /*Forced termination processing*/
    }

    /* ............ */
}

Note 1 When a task has already waited on the eventflag which has been created with TA_WSGL attribute (only one task is allowed to be in the WAITING state for the eventflag), this service call returns E_ILUSE error.

Note 2 Invoking tasks are queued to the target event flag (TA_WMUL attribute) wait queue in the order defined at creating the eventflag (FIFO order or current priority order). However, when the TA_CLR attribute is not specified, the wait queue is managed in the FIFO order even if the priority order is specified. This behavior falls outside µTRON4.0 specification.

Note 3 The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA_CLR attribute) is satisfied.

Note 4 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
- **pol_flg, ipol_flg** (Polling)

  This service call checks whether the bit pattern specified by parameter `waiptn` and the bit pattern that satisfies the required condition specified by parameter `wfmode` are set to the eventflag specified by parameter `flgid`.

  If the bit pattern that satisfies the required condition has been set to the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter `p_flgptn`.

  If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, “E_TMOUT” is returned.

  The following shows the specification format of required condition `wfmode`.

  - **wfmode = TWF_ANDW**
    Checks whether all of the bits to which 1 is set by parameter `waiptn` are set as the target eventflag.

  - **wfmode = TWF_ORW**
    Checks which bit, among bits to which 1 is set by parameter `waiptn`, is set as the target eventflag.

  The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf); /*Refer to note 3*/
void Task1 (VP_INT exinf)
{
    ER ercd; /*Declares variable*/
    ID flgid = 1; /*Declares and initializes variable*/
    FLGPTN waiptn = 14; /*Declares and initializes variable*/
    MODE wfmode = TWF_ANDW; /*Declares and initializes variable*/
    FLGPTN p_flgptn; /*Declares variable*/

    /* Wait for eventflag*/
    ercd = pol_flg (flgid, waiptn, wfmode, &p_flgptn);

    if (ercd == E_OK) {
        /* Polling success processing*/
    } else if (ercd == E_TMOUT) {
        /* Polling failure processing*/
    }

    /* ........... */
}
```

Note 1   When a task has already waited on the eventflag which has been created with TA_WSGL attribute (only one task is allowed to be in the WAITING state for the eventflag), this service call returns E_ILUSE error.

Note 2   The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA_CLR attribute) is satisfied.

Note 3   These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- twai_flg (Wait with time-out)

This service call checks whether the bit pattern specified by parameter waiptn and the bit pattern that satisfies the required condition specified by parameter wfmode are set to the eventflag specified by parameter flgid.

If a bit pattern that satisfies the required condition has been set for the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter p_flgptn.

If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue.

As a result, the invoking task is unlinked from the ready queue and is moved from the RUNNING state to the WAITING state (WAITING state for an eventflag).

The WAITING state for an eventflag is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for an Eventflag Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing set_flg.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing iset_flg.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_flg while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following shows the specification format of required condition wfmode.

- wfmode = TWF_ANDW
  Checks whether all of the bits to which 1 is set by parameter waiptn are set as the target eventflag.

- wfmode = TWF_ORW
  Checks which bit, among bits to which 1 is set by parameter waiptn, is set as the target eventflag.

The following describes an example for coding this service call.
Note 1 When a task has already waited on the eventflag which has been created with TA_WSGL attribute (only one task is allowed to be in the WAITING state for the eventflag), this service call returns E_ILUSE error.

Note 2 Invoking tasks are queued to the target event flag (TA_WMUL attribute) wait queue in the order defined at creating the eventflag (FIFO order or current priority order). However, when the TA_CLR attribute is not specified, the wait queue is managed in the FIFO order even if the priority order is specified. This behavior falls outside μITRON4.0 specification.

Note 3 The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA_CLR attribute) is satisfied.

Note 4 TMO_FEVR is specified for wait time tmout, processing equivalent to wai_flg will be executed. When TMO_POL is specified, processing equivalent to pol_flg will be executed.

Note 5 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);      /*Refer to note 5*/

void Task1 (VP_INT exinf)
{
   ER ercd;                   /*Declares variable*/
   ID flgid = 1;              /*Declares and initializes variable*/
   FLGPTN waiptn = 14;        /*Declares and initializes variable*/
   MODE wmode = TWF_ANDW;     /*Declares and initializes variable*/
   FLGPTN p_flgptn;           /*Declares variable*/
   TMO tmout = 3600;          /*Declares and initializes variable*/

   /* ......... */
   /*Wait for eventflag*/
   ercd = twai_flg (flgid, waiptn, wmode, &p_flgptn, tmout);

   if (ercd == E_OK) {
      /* ......... */ /*Normal termination processing*/
   } else if (ercd == E_RLWAI) {
      /* ......... */ /*Forced termination processing*/
   } else if (ercd == E_TMOUT) {
      /* ......... */ /*Time-out processing*/
   }

   /* ......... */
}
```
7.3.6  Reference eventflag state

An eventflag status is referenced by issuing the following service call from the processing program.

- ref_flg, iref_flg

Stores eventflag state packet (ID number of the task at the head of the wait queue, current bit pattern, etc.) of the eventflag specified by parameter flgid in the area specified by parameter pk_rflg.

The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1
void Task1 (VP_INT exinf);       /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ID flgid = 1;                  /*Declares and initializes variable*/
    T_RFLG pk_rflg;                /*Declares data structure*/
    ID wtskid;                    /*Declares variable*/
    FLGPTN flgptn;                /*Declares variable*/
    /* ......... */
    ref_flg (flgid, &pk_rflg);     /*Reference eventflag state*/
    wtskid = pk_rflg.wtskid;       /*Reference ID number of the task at the */
    /*head of the wait queue*/
    flgptn = pk_rflg.flgptn;       /*Reference current bit pattern*/
    /* ......... */
}
```

Note 1  For details about the eventflag state packet, refer to "[Eventflag state packet: T_RFLG]".

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.4 Data Queues

Multitask processing requires the inter-task communication function (data transfer function) that reports the processing result of a task to another task. The RI600PX therefore provides the data queues for transferring the prescribed size of data.

The following shows a processing flow when using a data queue.

![Figure 7-3 Processing Flow (Data Queue)](image)

Task A
Priority: High

Task B
Priority: Low

Receive from data queue

Reception wait period

Send to data queue

Note  Data units of 4 bytes are transmitted or received at a time.
7.4.1 Create data queue

Data queues are created by one of the following methods.

1) Creation by the system configuration file
   The static API “data_queue[]” described in the system configuration file creates a data queue. Refer to “20.13 Data Queue Information (dataqueue[])” for the details of “data_queue[]”.

2) Creation by cre_dtq or acre_dtq
   The cre_dtq creates a data queue with data queue ID indicated by parameter dtqid according to the content of parameter pk_cdtq.
   The acre_dtq creates a data queue according to the content of parameter pk_cdtq, and returns the created data queue ID.
   The information specified is shown below.

   - Data queue attribute (dtqatr)
     The following informations are specified as dtqatr:
     - The order of task wait queue for sending (FIFO order or task current priority order)
   - Capacity of the data queue area (dtqcnt)
     Start address of the data queue area (dtq)
     The TSZ_DTQ(dtqcnt) bytes area from the address indicated by parameter dtq is used for the data queue area. Refer to “18.3.2 Macros for Data Queue” for details of TSZ_DTQ macro.
     The data queue area should be generated to the area other than memory objects and user stacks.

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_dtq as a representative.

```c
#include "kernel.h"         /*Standard header file definition*/
#include "kernel_id.h"       /*Header file generated by cfg600px*/

#define DTQCNT 10             /*Capacity of the data queue area*/
/*(the number of data elements)*/

#pragma section B BRI_RAM   /*Section for the data queue area*/
static UW dtq_area[(TSZ_DTQ(dtqcnt)/sizeof(UW))]; /*Data queue area*/
#pragma section

#pragma task Task1          /*Refer to note*/
void Task1 (VP_INT exinf); /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ER dtqid;              /*Declares variable*/
    T_CDTQ pk_cdtq = {     /*Declares and initializes variable*/
        TA_TFIFO,         /*Data queue attribute (dtqatr)*/
        DTQCNT,          /*Capacity of the data queue area (dtqcnt)*/
        (VP)dtq_area     /*Start address of the data queue area (dtq)*/
    };
    /* ............ */

    dtqid = acre_dtq (&pk_cdtq); /*Create data queue*/
    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
7.4.2 Delete data queue

- del_dtq

This service call deletes the data queue specified by parameter dtqid.

When there are waiting tasks for the target data queue by using snd_dtq, tsnd_dtq, rcv_dtq or trcv_dtq, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the snd_dtq, tsnd_dtq, rcv_dtq or trcv_dtq.

This service call can be called from tasks that belong to Trusted Domain.

The following describes an example for coding this service call.

```
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note*/
void Task1 (VP_INT exinf);       /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ID dtqid = 8;          /*Declares and initializes variable*/
    /* .......... */
    ercd = del_dtq (dtqid); /*Delete data queue*/
    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

7.4.3 Send to data queue

A data is transmitted by issuing the following service call from the processing program.

- `snd_dtq` (Wait)
- `psnd_dtq`, `ipsnd_dtq` (Polling)
- `tsnd_dtq` (Wait with time-out)

- `snd_dtq` (Wait)
  This service call processes as follows according to the situation of the data queue specified by the parameter `dtqid`.
  - There is a task in the reception wait queue.
    This service call transfers the data specified by parameter `data` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  - There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.
    This service call stores the data specified by parameter `data` to the data queue.
  - There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.
    This service call queues the invoking task to the transmission wait queue of the target data queue and moves it from the RUNNING state to the WAITING state (data transmission wait state).

The sending WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>rcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>prcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>iprcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>trcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The data queue is reset as a result of issuing <code>vrst_dtq</code>.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_dtq</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1  Data is written to the data queue area in the order of the data transmission request.

Note 2  Invoking tasks are queued to the transmission wait queue of the target data queue in the order defined at creating the data queue (FIFO order or current priority order).

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1              /*Refer to note 3*/
void Task1 (VP_INT exinf);     /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ER ercd;                    /*Declares variable*/
    ID dtqid = 1;               /*Declares and initializes variable*/
    VP_INT data = 123;          /*Declares and initializes variable*/

    /* ........... */
    ercd = snd_dtq (dtqid, data);  /*Send to data queue*/
    if (ercd == E_OK) {
        /* ........... */        /*Normal termination processing*/
    } else if (ercd == E_RLWAI) {  /*Forced termination processing*/
        /* ........... */
    }
    /* ........... */
}
```
- **psnd_dtq, ipsnd_dtq (Polling)**
  These service calls process as follows according to the situation of the data queue specified by the parameter `dtqid`.

  - There is a task in the reception wait queue.
    These service calls transfer the data specified by parameter `data` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  - There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.
    These service calls store the data specified by parameter `data` to the data queue.

  - There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.
    These service calls return “E_TMOUT”.

The following describes an example for coding these service calls.

```c
#include "kernel.h"              /* Standard header file definition */
#include "kernel_id.h"           /* Header file generated by cfg600px */
#include "kernel_id.h"           /* Refer to note 2 */
#include "kernel_id.h"           /* Refer to note 2 */
void Task1 (VP_INT exinf);          /* Refer to note 2 */

void Task1 (VP_INT exinf)
{
  ER ercd;                   /* Declares variable */
  ID dtqid = 1;              /* Declares and initializes variable */
  VP_INT data = 123;         /* Declares and initializes variable */

  /* .......... */
  ercd = psnd_dtq (dtqid, data);  /* Send to data queue */

  if (ercd == E_OK) {
    /* .......... */             /* Polling success processing */
  } else if (ercd == E_TMOUT) {
    /* .......... */             /* Polling failure processing */
  }

  /* .......... */
}
```

**Note 1**  Data is written to the data queue area in the order of the data transmission request.

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.
- `tsnd_dtq` (Wait with time-out)
  This service call processes as follows according to the situation of the data queue specified by the parameter `dtqid`.

  - There is a task in the reception wait queue.
    This service call transfers the data specified by parameter `data` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  - There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.
    This service call stores the data specified by parameter `data` to the data queue.

  - There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.
    This service call queues the invoking task to the transmission wait queue of the target data queue and moves it from the RUNNING state to the WAITING state with time (data transmission wait state). The sending WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>rcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>prcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>iprcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>trcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The data queue is reset as a result of issuing <code>vrst_dtq</code>.</td>
<td>E_RST</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_dtq</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task

void Task1 (VP_INT exinf);          /*Refer to note 4*/

void Task1 (VP_INT exinf)
{
    ER     ercd;                   /*Declares variable*/
    ID     dtqid = 1;              /*Declares and initializes variable*/
    VP_INT data = 123;             /*Declares and initializes variable*/
    TMO    tmout = 3600;           /*Declares and initializes variable*/

    /* ........ */
    /*Send to data queue*/
    ercd = tsnd_dtq (dtqid, data, tmout);

    if (ercd == E_OK) {
        /* ........ */
    } else if (ercd == E_RLWA1) {
        /* ........ */
    } else if (ercd == E_TMOUT) {
        /* ........ */
    }

    /* ........ */
}
```

Note 1 Data is written to the data queue area in the order of the data transmission request.

Note 2 Invoking tasks are queued to the transmission wait queue of the target data queue in the order defined at creating the data queue (FIFO order or current priority order).

Note 3 TMO_FEVR is specified for wait time tmout, processing equivalent to snd_dtq will be executed. When TMO_POL is specified, processing equivalent to psnd_dtq will be executed.

Note 4 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

Note 1 Data is written to the data queue area in the order of the data transmission request.

Note 2 Invoking tasks are queued to the transmission wait queue of the target data queue in the order defined at creating the data queue (FIFO order or current priority order).

Note 3 TMO_FEVR is specified for wait time tmout, processing equivalent to snd_dtq will be executed. When TMO_POL is specified, processing equivalent to psnd_dtq will be executed.

Note 4 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
7.4.4 Forced send to data queue

Data is forcibly transmitted by issuing the following service call from the processing program.

- **fsnd_dtq**, **fsnd_dtq**
  
  This service call processes as follows according to the situation of the data queue specified by the parameter *dtqid*.

  - There is a task in the reception wait queue.
    
    This service call transfers the data specified by parameter *data* to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  
  - There is no task neither in the reception wait queue and transmission wait queue.
    
    This service call stores the data specified by parameter *data* to the data queue. If there is no available space in the data queue, this service call deletes the oldest data in the data queue before storing the data specified by *data* to the data queue.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note 2*/
void    Task1 (VP_INT  exinf);          /*Refer to note 2*/
void    Task1 (VP_INT  exinf)
{
    ID    dtqid = 1;              /*Declares and initializes variable*/
    VP_INT  data = 123;             /*Declares and initializes variable*/
    /* ......... */
    fsnd_dtq (dtqid, data);         /*Forced send to data queue*/
    /* ......... */
}
```

Note 1  Data is written to the data queue area in the order of the data transmission request.

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
7.4.5 Receive from data queue

A data is received (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- rcv_dtq (Wait)
- prcv_dtq, iprcv_dtq (Polling)
- trcv_dtq (Wait with time-out)

- rcv_dtq (Wait)

This service call processes as follows according to the situation of the data queue specified by the parameter dtqid.

- There is a data in the data queue.
  This service call takes out the oldest data from the data queue and stores the data to the area specified by p_data.
  When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.

- There is no data in the data queue and there is a task in the transmission wait queue.
  This service call stores the data specified by the task in the top of the transmission wait queue to the area specified by p_data. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  Note, this situation is caused only when the capacity of the data queue is 0.

- There is no data in the data queue and there is no task in the transmission wait queue.
  This service call queues the invoking task to the reception wait queue of the target data queue and moves it from the RUNNING state to the WAITING state (data reception wait state).
  The receiving WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data was sent to the data queue area as a result of issuing snd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing psnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing ipsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing tsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing fsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing ifsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_dtq while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1 Invoking tasks are queued to the reception wait queue of the target data queue in the order of the data reception request.

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

```c
#include "kernel.h"             /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1               /*Refer to note 2*/
void Task1 (VP_INT exinf);      /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;                      /*Declares variable*/
    ID dtqid = 1;                 /*Declares and initializes variable*/
    VP_INT p_data;                /*Declares variable*/

    /* ........... */
    ercd = rcv_dtq (dtqid, &p_data);

    if (ercd == E_OK) {
        /* ........... */            /*Normal termination processing*/
    } else if (ercd == E_RLWAI) {
        /* ........... */            /*Forced termination processing*/
    }

    /* ........... */
}
```
- prcv_dtq, iprcv_dtq (Polling)
  These service calls process as follows according to the situation of the data queue specified by the parameter dtqid.
  - There is a data in the data queue.
    This service call takes out the oldest data from the data queue and stores the data to the area specified by p_data.
    When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.
  - There is no data in the data queue and there is a task in the transmission wait queue.
    These service calls store the data specified by the task in the top of the transmission wait queue to the area specified by p_data. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
    Note, this situation is caused only when the capacity of the data queue is 0.
  - There is no data in the data queue and there is no task in the transmission wait queue.
    These service calls return "E_TMOUT".

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/

void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
  ER ercd;                   /*Declares variable*/
  ID dtqid = 1;              /*Declares and initializes variable*/
  VP_INT p_data;             /*Declares variable*/

  /* ......... */               /*Receive from data queue*/

  ercd = prcv_dtq (dtqid, &p_data);

  if (ercd == E_OK) {
    /* ......... */               /*Polling success processing*/
  } else if (ercd == E_TMOUT) {
    /* ......... */               /*Polling failure processing*/
  }

  /* ......... */
}
```

**Note** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
- **trcv_dtq** (Wait with time-out)
  This service call processes as follows according to the situation of the data queue specified by the parameter *dtqid*.

  - There is a data in the data queue.
    This service call takes out the oldest data from the data queue and stores the data to the area specified by *p_data*.
    When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.

  - There is no data in the data queue and there is a task in the transmission wait queue.
    This service call stores the data specified by the task in the top of the transmission wait queue to the area specified by *p_data*. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
    Note, this situation is caused only when the capacity of the data queue is 0.

  - There is no data in the data queue and there is no task in the transmission wait queue.
    This service call queues the invoking task to the reception wait queue of the target data queue and moves it from the RUNNING state to the WAITING state with time-out (data reception wait state).
    The receiving WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>snd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>psnd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>ipsnd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>tsnd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>fsnd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <em>ifsnd_dtq</em>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <em>rel_wai</em> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <em>irel_wai</em> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by <em>tmout</em> has elapsed.</td>
<td>E_TMOOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept <em>del_dtq</em> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1  Invoking tasks are queued to the reception wait queue of the target data queue in the order of the data reception request.

Note 2  TMO_FEVR is specified for wait time \texttt{tmout}, processing equivalent to \texttt{rcv_dtq} will be executed. When TMO_POL is specified, processing equivalent to \texttt{prcv_dtq} will be executed.

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the \texttt{cfg600px} generates these statement into the \texttt{“kernel_id.h”}.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1                   /*Refer to note 3*/
void Task1 (VP_INT exinf);          /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ER ercd;               /*Declares variable*/
    ID dtqid = 1;          /*Declares and initializes variable*/
    VP_INT p_data;         /*Declares variable*/
    TMO tmout = 3600;      /*Declares and initializes variable*/

    /*Receive from data queue*/
    ercd = trcv_dtq (dtqid, &p_data, tmout);

    if (ercd == E_OK) {
        /*Normal termination processing*/
    } else if (ercd == E_RLWA1) {
        /*Forced termination processing*/
    } else if (ercd == E_TMOUT) {
        /*Time-out processing*/
    }

    /*.............*/
}
```
### 7.4.6 Reference data queue state

A data queue status is referenced by issuing the following service call from the processing program.

```c
#define "kernel.h" /*Standard header file definition*/
#define "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf);
    /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID dtqid = 1; /*Declares and initializes variable*/
    T_RDTQ pk_rdtq; /*Declares data structure*/
    ID stskid; /*Declares variable*/
    ID rtskid; /*Declares variable*/
    UINT sdtqcnt; /*Declares variable*/

    /* .......... */

    ref_dtq (dtqid, &pk_rdtq); /*Reference data queue state*/

    stskid = pk_rdtq.stskid; /*Acquires existence of tasks waiting for */
                        /*data transmission*/
    rtskid = pk_rdtq.rtskid; /*Acquires existence of tasks waiting for */
                        /*data reception*/
    sdtqcnt = pk_rdtq.sdtqcnt; /*Reference the number of data elements in */
                        /*data queue*/

    /* .......... */
}
```

**Note 1**  For details about the data queue state packet, refer to "[Data queue state packet: T_RDTQ]".

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.5 Mailboxes

Multitask processing requires the inter-task communication function (message transfer function) that reports the processing result of a task to another task. The RI600PX therefore provides the mailbox for transferring the start address of a message written in the shared memory area.

The following shows a processing flow when using a mailbox.

![Figure 7-4 Processing Flow (Mailbox)]

7.5.1 Messages

The information exchanged among processing programs via the mailbox is called “messages”. Messages can be transmitted to any processing program via the mailbox, but it should be noted that, in the case of the synchronization and communication functions of the RI600PX, only the start address of the message is handed over to the receiving processing program, but the message contents are not copied to a separate area.

- Message area
  The message must be generated in the memory objects that both transmitting task and receiving task can access. However, the management table exists in the top of message area. The system operation cannot be guaranteed if the management table is destroyed. For this reason, data queue or message buffer is recommended for message communication.
- Basic form of messages
  In the RI600PX, the message contents and length are prescribed as follows, according to the attributes of the mailbox to be used.

- When using a mailbox with the TA_MFIFO attribute
  The message must be started from the T_MSG structure. This area is used by the kernel. The use message should be arranged following the T_MSG structure.
  The length of the message is prescribed among the processing programs that exchange data using the mailbox.
  The following shows the basic form of coding TA_MFIFO attribute messages.

  [Message packet for TA_MFIFO attribute]

```c
/* T_MSG structure, which is defined in the kernel.h*/
typedef struct {
    VP  msghead;       /*RI600PX management area*/
} T_MSG;

/* Message structure defined by user*/
typedef struct {
    T_MSG  t_msg;       /*T_MSG structure*/
    B  data[8];       /*User message*/
} USER_MSG;
```

- When using a mailbox with the TA_MPRI attribute
  The message must be started from the T_MSG_PRI structure. The T_MSG_PRI.msgque is used by the kernel.
  The message priority should be set to T_MSG_PRI.msgpri.
  The length of the message is prescribed among the processing programs that exchange data using the mailbox.
  The following shows the basic form of coding TA_MPRI attribute messages.

  [Message packet for TA_MPRI attribute]

```c
/* T_MSG structure, which is defined in the kernel.h*/
typedef struct {
    VP  msghead;       /*RI600PX management area*/
} T_MSG;

/* T_MSG_PRI structure, which is defined in the kernel.h*/
typedef struct {
    T_MSG  msgque;        /*Message header*/
    PRI  msgpri;        /*Message priority*/
} T_MSG_PRI;

/* Message structure defined by user*/
typedef struct {
    T_MSG_PRI  t_msg;       /*T_MSG_PRI structure*/
    B  data[8];       /*User message*/
} USER_MSG;
```

Note 1  In the RI600PX, a message having a smaller priority number is given a higher priority.

Note 2  Values that can be specified as the message priority level are limited to the range defined by Maximum message priority (max_pri) in Mailbox Information (mailbox[]) when the system configuration file is created.
7.5.2 Create mailbox

Mailboxes are created by one of the following methods.

1) Creation by the system configuration file
   The static API "mailbox[]" described in the system configuration file creates a mailbox.
   Refer to "20.14 Mailbox Information (mailbox[])" for the details of "mailbox[]".

2) Creation by cre_mbx or acre_mbx
   The cre_mbx creates a mailbox with mailbox ID indicated by parameter mbxid according to the content of parameter pk_cmbx.
   The acre_mbx creates a mailbox according to the content of parameter pk_cmbx, and returns the created mailbox ID.
   The information specified is shown below.
   - Mailbox attribute (mbxatr)
     The following informations are specified as mbxatr:
     - The order of task wait queue (FIFO order or task current priority order)
     - The order of message queue (FIFO order or message priority order)
   - Maximum message priority (maxmpri)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_mbx as a representative.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER mbxid;                        /*Declares variable*/
    T_CMBX pk_cmbx = {               /*Declares and initializes variable*/
        TA_TFIFO|TA_MFIFO,          /*Mailbox attribute (mbxatr)*/
        1,                          /*Maximum message priority (maxmpri)*/
        0                           /*Reserved (mprihd)*/
    };
    /* ............ */

    mbxid = acre_mbx ( &pk_cmbx );  /*Create mailbox/ */
    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.5.3 Delete mailbox

- del_mbx

This service call deletes the mailbox specified by parameter mbxid.
When there are waiting tasks for the target mailbox by using rcv_mbx or trcv_mbx, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the rcv_mbx or trcv_mbx.
This service call can be called from tasks that belong to Trusted Domain.
The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/
void Task1 (VP_INT exinf);         /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID mbxid = 8;          /*Declares and initializes variable*/
    /* ......... */
    ercd = del_mbx ( mbxid );   /*Delete semaphore*/
    /* ......... */
}
```

Note  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

7.5.4 Send to mailbox

A message is transmitted by issuing the following service call from the processing program.

- snd_mbx, isnd_mbx

This service call transmits the message specified by parameter pk_msg to the mailbox specified by parameter mbxid (queues the message in the wait queue).

If a task is queued to the target mailbox wait queue when this service call is issued, the message is not queued but handed over to the relevant task (first task of the wait queue).

As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (receiving WAITING state for a mailbox) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 3*/

void Task1 (VP_INT exinf);
/*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ID mbxid = 1; /*Declares and initializes variable*/
    T_MSG_PRI *pk_msg; /*Declares data structure*/

    /* ........ */
    
    /* ........ */ /*Secures memory area (for message)*/
    pk_msg = ... /* and set the pointer to pk_msg*/
    /* ........ */ /*Creates message (contents)*/

    pk_msg->msgpri = 8; /*Initializes data structure*/

    /*Send to mailbox*/
    snd_mbx (mbxid, (T_MSG *) pk_msg);

    /* ........ */
}
```

Note 1 Messages are queued to the target mailbox in the order defined by queuing method during configuration (FIFO order or message priority order).

Note 2 For details about the message packet T_MSG and T_MSG_PRI, refer to "7.5.1 Messages".

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
7.5.5 Receive from mailbox

A message is received (infinite wait, polling, or with time-out) by issuing the following service call from the processing program.

- `rcv_mbx` (Wait)
- `prcv_mbx`, `iprcv_mbx` (Polling)
- `trcv_mbx` (Wait with time-out)

- `rcv_mbx` (Wait)

This service call receives a message from the mailbox specified by parameter `mbxid`, and stores its start address in the area specified by parameter `ppk_msg`.

If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service call does not receive messages but queues the invoking task to the target mailbox wait queue and moves it from the RUNNING state to the WAITING state (message reception wait state).

The receiving WAITING state for a mailbox is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Mailbox Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>snd_mbx</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>isnd_mbx</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbx</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note 3*/
void Task1 (VP_INT exinf)          /*Refer to note 3*/
{
    ER ercd;                   /*Declares variable*/
    ID mbxid = 1;              /*Declares and initializes variable*/
    T_MSG *ppk_msg;            /*Declares data structure*/
    /* ......... */
    /*Receive from mailbox*/
    ercd = rcv_mbx (mbxid, &ppk_msg);
    if (ercd == E_OK) {
        /* ......... */         /*Normal termination processing*/
    } else if (ercd == E_RLWAI) {
        /* ......... */         /*Forced termination processing*/
    }
    /* ......... */
}
```

Note 1 Invoking tasks are queued to the target mailbox wait queue in the order defined at creating the mailbox (FIFO order or current priority order).

Note 2 For details about the message packet `T_MSG` and `T_MSG_PRI`, refer to "7.5.1 Messages".

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
- prcv_mbx, iprcv_mbx (Polling)

This service call receives a message from the mailbox specified by parameter \textit{mbxid}, and stores its start address in the area specified by parameter \textit{ppk_msg}.

If the message could not be received from the target mailbox (no messages were queued in the wait queue) when this service call is issued, message reception processing is not executed but “E_TMOUT” is returned.

The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
#pragma task Task1              /*Refer to note 2*/
void Task1 (VP_INT exinf);      /*Refer to note 2*/
void Task1 (VP_INT exinf) {
    ER ercd;                      /*Declares variable*/
    ID mbxid = 1;                 /*Declares and initializes variable*/
    T_MSG *ppk_msg;               /*Declares data structure*/

    /* ............. */            /*Receive from mailbox*/
    ercd = prcv_mbx (mbxid, &ppk_msg);

    if (ercd == E_OK) {
        /* ............. */        /*Polling success processing*/
    } else if (ercd == E_TMOUT) {
        /* ............. */        /*Polling failure processing*/
    }

    /* ............. */
}
```

**Note 1** For details about the message packet T_MSG and T_MSG_PRI, refer to “7.5.1 Messages”.

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- **trcv_mbx** (Wait with time-out)
  This service call receives a message from the mailbox specified by parameter `mbxid`, and stores its start address in the area specified by parameter `ppk_msg`.
  If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service call does not receive messages but queues the invoking task to the target mailbox wait queue and moves it from the RUNNING state to the WAITING state with time-out (message reception wait state). The receiving WAITING state for a mailbox is cancelled in the following cases.

### Receiving WAITING State for a Mailbox Cancel Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>snd_mbx</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>isnd_mbx</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td><code>E_TMOOUT</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbx</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include "kernel.h"        /*Standard header file definition*/
#include "kernel_id.h"      /*Header file generated by cfg600px*/
#pragma task Task1          /*Refer to note 4*/
void Task1 (VP_INT exinf); /*Refer to note 4*/

void Task1 (VP_INT exinf)
{
  ER  ercd;                  /*Declares variable*/
  ID  mbxid = 1;             /*Declares and initializes variable*/
  T_MSG *ppk_msg;            /*Declares data structure*/
  TMO tmout = 3600;          /*Declares and initializes variable*/

  /* ......... */
  ercd = trcv_mbx (mbxid, &ppk_msg, tmout);

  if (ercd == E_OK) {
    /* ......... */
  } else if (ercd == E_RLWAI) {
    /* ......... */
  } else if (ercd == E_TMOOUT) {
    /* ......... */
  }
  /* ......... */
}
```

**Note 1**  Invoking tasks are queued to the target mailbox wait queue in the order defined at creating the mailbox (FIFO order or current priority order).

**Note 2**  `TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `rcv_mbx` will be executed. When `TMO_POL` is specified, processing equivalent to `prcv_mbx` will be executed.

**Note 3**  For details about the message packet `T_MSG` and `T_MSG_PRI`, refer to “7.5.1 Messages”.

**Note 4**  These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”. 

7.5.6 Reference mailbox state

A mailbox status is referenced by issuing the following service call from the processing program.

- ref_mbx, iref_mbx

Stores mailbox state packet (ID number of the task at the head of the wait queue, start address of the message packet at the head of the wait queue) of the mailbox specified by parameter mbxid in the area specified by parameter pk_rmbx.

The following describes an example for coding these service calls.

```c
#include "kernel.h"  /* Standard header file definition */
#include "kernel_id.h" /* Header file generated by cfg600px */
#pragma task Task1 /* Refer to note 2 */
void Task1 (VP_INT exinf); /* Refer to note 2 */

void Task1 (VP_INT exinf)
{
    ID mbxid = 1;             /* Declares and initializes variable */
    T_RMBX pk_rmbx;           /* Declares data structure */
    ID wtskid;                /* Declares variable */
    T_MSG *pk_msg;            /* Declares data structure */

    /* .......... */

    ref_mbx (mbxid, &pk_rmbx); /* Reference mailbox state */

    wtskid = pk_rmbx.wtskid;   /* Reference ID number of the task at the */
                                /* head of the wait queue */
    pk_msg = pk_rmbx.pk_msg;   /* Reference start address of the message */
                                /* packet at the head of the wait queue */

    /* .......... */
}
```

Note 1 For details about the mailbox state packet, refer to "[Mailbox state packet: T_RMBX]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
CHAPTER 8 EXTENDED SYNCHRONIZATION AND COMMUNICATION FUNCTIONS

This chapter describes the extended synchronization and communication functions performed by the RI600PX.

8.1 Outline

The extended synchronization and communication function of the RI600PX provides Mutexes for implementing exclusive control between tasks, and Message Buffers for transferring messages of arbitrary size by copying the message.

8.2 Mutexes

Multitask processing requires the function to prevent contentions on using the limited number of resources (A/D converter, coprocessor, files, or the like) simultaneously by tasks operating in parallel (exclusive control function). To resolve such problems, the RI600PX therefore provides "mutexes".

The following shows a processing flow when using a mutex.

Figure 8-1 Processing Flow (Mutex)
8.2.1 Priority inversion problem

When a semaphore is used for exclusive control of a resource, a problem called priority inversion may arise. This refers to the situation where a task that is not using a resource delays the execution of a task requesting the resource. Figure 8-2 illustrates this problem. In this figure, tasks A and C are using the same resource, which task B does not use. Task A attempts to acquire a semaphore so that it can use the resource but enters the WAITING state because task C is already using the resource. Task B has a priority higher than task C and lower than task A. Thus, if task B is executed before task C has released the semaphore, release of the semaphore is delayed by the execution of task B. This also delays acquisition of the semaphore by task A. From the viewpoint of task A, a lower-priority task that is not even competing for the resource gets priority over task A.

To avoid this problem, use a mutex instead of a semaphore.

8.2.2 Current priority and base priority

A task has two priority levels: base priority and current priority. Tasks are scheduled according to current priority. While a task does not have a mutex locked, its current priority is always the same as its base priority. When a task locks a mutex, only its current priority is raised to the ceiling priority of the mutex.

When priority-changing service call chg_pri or ichg_pri is issued, both the base priority and current priority are changed if the specified task does not have a mutex locked. When the specified task locks a mutex, only the base priority is changed. When the specified task has a mutex locked or is waiting to lock a mutex, these service calls returns "E_ILUSE" if a priority higher than the ceiling priority of the mutex is specified.

The current priority can be checked through service call get_pri or iget_pri. And both the current priority and base priority can be referred by ref_tsk or iref_tsk.

8.2.3 Simplified priority ceiling protocol

Original behavior of the priority ceiling protocol is to make the current priority of the task to the highest ceiling priority of mutexes which are locked by the task. This behavior is achieved by controlling the current priority of the task as follows.

- When a task locks a mutex, changes the current priority of the task to the highest ceiling priority of mutexes which are locked by the task.
- When a task unlocks a mutex, changes the current priority of the task to the highest ceiling priority of mutexes which continues to be locked by the task. When there is no mutex locked by the task after unlock, returns the current priority of the task to the base priority.

However, the RI600PX adopts simplified priority ceiling protocol because of reducing overhead. Therefore, the underlined part is not processed.
8.2.4 Differences from semaphores

The mutex operates similarly to semaphores (binary semaphore) whose the maximum resource count is 1, but they differ in the following points.

- The current priority of the task which locks a mutex raises to the ceiling priority of the mutex until the task unlocks the mutex. As a result, the priority inversion problem is evaded.
  --> The current priority is not changed by using semaphore.

- A locked mutex can be unlocked (equivalent to returning of resources) only by the task that locked the mutex
  --> Semaphores can return resources via any task and handler.

- Unlocking is automatically performed when a task that locked the mutex is terminated (ext_tsk or ter_tsk)
  --> Semaphores do not return resources automatically, so they end with resources acquired.

- Semaphores can manage multiple resources (the maximum resource count can be assigned), but the maximum number of resources assigned to a mutex is fixed to 1.
8.2.5  Create mutex

Mutexes are created by one of the following methods.

1) Creation by the system configuration file

The static API *"mutex[]"* described in the system configuration file creates a mutex.
Refer to *"20.15 Mutex Information (mutex[])"* for the details of "mutex[]."

2) Creation by *cre_mtx* or *acre_mtx*

The *cre_mtx* creates a mutex with mutex ID indicated by parameter *mtxid* according to the content of parameter *pk_cmtx*.

The *acre_mtx* creates a mutex according to the content of parameter *pk_cmtx*, and returns the created mutex ID.

The information specified is shown below.

- Mutex attribute (*mtxatr*)
  Only TA_CEILING (Priority ceiling protocol) can be specified for *mtxatr*.
  Note, task wait queue is managed in task current priority order.

- Ceiling priority (*ceilpri*)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding *acre_mtx* as a representative.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
  ER     mtxid;              /*Declares variable*/
  T_CMTX pk_cmtx = {         /*Declares and initializes variable*/
    TA_CEILING,                 /*Mutex attribute (mtxatr)*/
    1                           /*Ceiling priority (ceilpri)*/
  };

  mtxid = acre_mtx ( &pk_cmtx );  /*Create mutex*/

  /* ........... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
8.2.6 Delete mutex

This service call deletes the mutex specified by parameter \textit{ mtxid}. When either of task locks the target mutex, the lock by the task is cancelled. As a result, the current task priority of the task is returned to the base priority when there is no mutex being locked by the task. The task is not notified that the mutex has been deleted. If an attempt is later made to unlock the mutex by using \textit{ unl_mtx }, an error \textit{ E_NOEXS } is returned.

When there are waiting tasks for the target mutex by using \textit{ loc_mtx } or \textit{ tloc_mtx }, this service call cancels the WAITING state of the tasks and returns \textit{ E_DLT } as a return value of the \textit{ loc_mtx } or \textit{ tloc_mtx }.

This service call can be called from tasks that belong to Trusted Domain.

The following describes an example for coding this service call.

```c
#include "kernel.h"    /*Standard header file definition*/
#include "kernel_id.h"  /*Header file generated by cfg600px*/
#pragma task Task1     /*Refer to note*/
void Task1 (VP_INT exinf); /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID mtxid = 8;          /*Declares and initializes variable*/
    /* ............ */
    ercd = del_mtx ( mtxid ); /*Delete semaphore*/
    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the \textit{cfg600px} generates these statement into the "kernel_id.h".
8.2.7 Lock mutex

Mutexes can be locked by issuing the following service call from the processing program.

- `loc_mtx` (Wait)
- `ploc_mtx` (Polling)
- `tloc_mtx` (Wait with time-out)

- `loc_mtx` (Wait)
  
  This service call locks the mutex specified by parameter `mtxid`.
  
  If the target mutex could not be locked (another task has been locked) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to the WAITING state (mutex wait state).

  The WAITING state for a mutex is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Mutex Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>unl_mtx</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>ext_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>exd_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>ter_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mtx</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the invoking task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

The following describes an example for coding this service call.

```c
#include    "kernel.h" /*Standard header file definition*/
#include    "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 3*/
void Task1 (VP_INT exinf); /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ER ercd; /* Declares variable */
    ID mtxid = 8; /* Declares and initializes variable */
    /* ... */
    ercd = loc_mtx (mtxid); /* Lock mutex */
    if (ercd == E_OK) {
        /* ... */
        /* Locked state */
        unl_mtx (mtxid); /* Unlock mutex */
    } else if (ercd == E_RLWAI) {
        /* ... */
        /* Forced termination processing */
    }
    /* ... */
}
```
Note 1  Invoking tasks are queued to the target mutex wait queue in the priority order. Among tasks with the same priority, they are queued in FIFO order.

Note 2  This service call returns “E_ILUSE” if this service call is re-issued for the mutex that has been locked by the invoking task (multiple-locking of mutex).

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- **ploc_mtx (Polling)**

   This service call locks the mutex specified by parameter `mtxid`. 

   If the target mutex could not be locked (another task has been locked) when this service call is issued but 
   "E_TMOUT" is returned. 

   When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the 
   target mutex. However, the this service call does not change the current priority when the invoking task has locked 
   other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked 
   mutexes. 

   The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                    /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;                   /*Declares variable*/
    ID mtxid = 8;              /*Declares and initializes variable*/

    /* ........... */

    ercd = ploc_mtx (mtxid);        /*Lock mutex*/

    if (ercd == E_OK) {
        /* ........... */

        unl_mtx (mtxid);            /*Unlock mutex*/

    } else if (ercd == E_TMOUT) {
        /* ........... */

        /* Polling failure processing*/
    }

    /* ........... */
}
```

**Note 1**  This service call returns "E_ILUSE" if this service call is re-issued for the mutex that has been locked by the 
invoking task (multiple-locking of mutex).

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the 
cfg600px generates these statement into the "kernel_id.h".
- **tloc_mtx** (Wait with time-out)

This service call locks the mutex specified by parameter `mtxid`. If the target mutex could not be locked (another task has been locked) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to the WAITING state with time-out (mutex wait state).

The WAITING state for a mutex is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Mutex Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>unl_mtx</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>ext_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>exd_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing <code>ter_tsk</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td><code>E_TMOOUT</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mtx</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the target mutex. However, the this service call does not change the current priority when the invoking task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

The following describes an example for coding this service call.

```c
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h"  /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note 4*/
void Task1 (VP_INT exinf);  /*Refer to note 4*/

void Task1 (VP_INT exinf)
{
  ER ercd;  /*Declares variable*/
  ID mtxid = 8;  /*Declares and initializes variable*/
  TMO tmout = 3600;  /*Declares and initializes variable*/

  /* .......... */
  ercd = tloc_mtx (mtxid, tmout);  /*Lock mutex*/
  if (ercd == E_OK) {
    /* .......... */  /*Locked state*/
    unl_mtx (mtxid);  /*Unlock mutex*/
  } else if (ercd == E_RLWAI) {
    /* .......... */  /*Forced termination processing*/
  } else if (ercd == E_TMOOUT) {
    /* .......... */  /*Time-out processing*/
  }
  /* .......... */
}
```

**Note 1** Invoking tasks are queued to the target mutex wait queue in the priority order. Among tasks with the same priority, they are queued in FIFO order.

**Note 2** This service call returns "E_ILUSE" if this service call is re-issued for the mutex that has been locked by the invoking task (multiple-locking of mutex).

**Note 3** `TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `loc_mtx` will be executed. When `TMO_POL` is specified, processing equivalent to `ploc_mtx` will be executed.
Note 4  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
8.2.8 Unlock mutex

The mutex locked state can be cancelled by issuing the following service call from the processing program.

- `unl_mtx`
  This service call unlocks the locked mutex specified by parameter `mtxid`.
  If a task has been queued to the target mutex wait queue when this service call is issued, mutex lock processing is performed by the task (the first task in the wait queue) immediately after mutex unlock processing.
  As a result, the task is unlinked from the wait queue and moves from the WAITING state (mutex wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. And this service call changes the current priority of the task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

The following describes an example for coding this service call.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/

#pragma task TaskI /*Refer to note 3*/
void TaskI (VP_INT exinf); /*Refer to note 3*/

void TaskI (VP_INT exinf)
{
    ER ercd; /*Declares variable*/
    ID mtxid = 8; /*Declares and initializes variable*/

    /* ......... */
    ercd = loc_mtx (mtxid); /*Lock mutex*/

    if (ercd == E_OK) {
        /* ......... */ /*Locked state*/
        unl_mtx (mtxid); /*Unlock mutex*/
    } else if (ercd == E_RLWAI) {
        /* ......... */ /*Forced termination processing*/
    }

    /* ......... */
}
```

Note 1 A locked mutex can be unlocked only by the task that locked the mutex.
If this service call is issued for a mutex that was not locked by the invoking task, no processing is performed but "E_ILUSE" is returned.

Note 2 When terminating a task, the mutexes which are locked by the terminated task are unlocked.

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
8.2.9 Reference mutex state

A mutex status is referenced by issuing the following service call from the processing program.

- ref_mtx.

This service call stores the detailed information of the mutex specified by parameter mtxid (existence of locked mutexes, waiting tasks, etc.) into the area specified by parameter pk_mtx.

The following describes an example for coding this service call.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1
void Task1 (VP_INT exinf);          /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ID mtxid = 1;              /*Declares and initializes variable*/
    T_RMTX pk_rmtx;                /*Declares data structure*/
    ID htskid;                 /*Declares variable*/
    ID wtskid;                 /*Declares variable*/

    /* ......... */

    ref_mtx (mbxid, &pk_rmtx);      /*Reference mutex state*/

    htskid = pk_rmtx.htskid;        /*Acquires existence of locked mutexes*/
    wtskid = pk_rmtx.wtskid;        /*Reference ID number of the task at the */
    /*head of the wait queue*/

    /* ......... */
}
```

Note 1 For details about the mutex state packet, refer to "[Mutex state packet: T_RMTX]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
8.3 Message Buffers

Multitask processing requires the inter-task communication function (message transfer function) that reports the processing result of a task to another task. The RI600PX therefore provides the message buffers for copying and transferring the arbitrary size of message.

The following shows a processing flow when using a message buffer.

Figure 8-3 Processing Flow (Message buffer)

Task A
Priority: High

Task B
Priority: Low

Receive from message buffer

Reception wait period

Send to message buffer
8.3.1 Create message buffer

Message buffers are created by one of the following methods.

1) Creation by the system configuration file
   The static API "message_buffer[]" described in the system configuration file creates a message buffer.
   Refer to "20.16 Message Buffer Information (message_buffer[])" for the details of "message_buffer[]".

2) Creation by cre_mbf or acre_mbf
   The cre_mbf creates a message buffer with message buffer ID indicated by parameter mbfid according to the
   content of parameter pk_cmbf.
   The acre_mbf creates a message buffer according to the content of parameter pk_cmbf, and returns the created
   message buffer ID.
   The information specified is shown below.
   - Message buffer attribute (mbfatr)
     Only TA_TFIFO (the order of task wait queue for sending is managed by FIFO order.) can be specified for
     mbfatr.
   - Maximum message size (maxmsz)
   - Size of the message buffer area (mbfsz). Start address of the message buffer area (mbf)
     The message buffer area should be generated to the area other than memory objects and user stacks.

These service calls can be called from tasks that belong to Trusted Domain.

The following describes an example for coding acre_mbf as a representative.

```c
#include    "kernel.h"    /*Standard header file definition*/
#include    "kernel_id.h"  /*Header file generated by cfg600px*/
#define MAX_MSGSZ 64        /*Maximum message size (in bytes)*/
#define MBFSZ     256        /*Size of the message buffer area (in bytes)*/
#pragma section B BRI_RAM  /*Section for the message buffer area*/
static UW mbf_area[MBFSZ/sizeof(UW)]; /*Message buffer area*/
#pragma section
#pragma task Task1          /*Refer to note*/
void Task1 (VP_INT exinf); /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER mbfid;                  /*Declares variable*/
    T_CMBF pk_cmbf = {         /*Declares and initializes variable*/
        TA_TFIFO,            /*Message buffer attribute (mbfatr)*/
        MAX_MSGSZ,          /*Maximum message size (in bytes) (maxmsz)*/
        MBFSZ,              /*Size of the message buffer area (in bytes) (mbfsz)*/
        (VP)mbf_area        /*Start address of the message buffer area (mbf)*/
    };
    /* .......... */
    mbfid = acre_mbf (&pk_cmbf); /*Create message buffer*/
    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the
 cfg600px generates these statement into the "kernel_id.h".
8.3.2 Delete message buffer

- del_mbf

This service call deletes the message buffer specified by parameter mbfid.
When there are waiting tasks for the target message buffer by using snd_mbf, tsnd_mbf, rcv_mbf or trcv_mbf, this
service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the snd_mbf, tsnd_mbf,
rcv_mbf or trcv_mbf.
This service call can be called from tasks that belong to Trusted Domain.
The following describes an example for coding this service call.

```
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1              /*Refer to note*/
void Task1 (VP_INT exinf);      /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID mbfid = 8;          /*Declares and initializes variable*/
    /* ........... */
    ercd = del_mbf (mbfid);   /*Delete data queue*/
    /* ........... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the
cfg600px generates these statement into the "kernel_id.h".
### 8.3.3 Send to message buffer

A message is transmitted by issuing the following service call from the processing program.

- `snd_mbf` (Wait)
- `psnd_mbf`, `ipsnd_mbf` (Polling)
- `tsnd_mbf` (Wait with time-out)

This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

- There is a task in the reception wait queue. This service call transfers the message specified by parameter `msg` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer. This service call stores the message specified by parameter `msg` to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  \[
  \text{The amount of decrease} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL}
  \]

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue. This service call queues the invoking task to the transmission wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state (message transmission wait state). The sending WAITING state for a message buffer is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing <code>rcv_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing <code>prcv_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing <code>trcv_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcedly released from waiting by following either.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td></td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td></td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>ter_tsk</code> while waiting).</td>
<td></td>
</tr>
<tr>
<td>- The time specified by <code>tmout</code> for <code>tsnd_mbf</code> has elapsed.</td>
<td></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>The message buffer is reset as a result of issuing <code>vrst_mbf</code>.</td>
<td><code>EV_RST</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbf</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
CHAPTER 8  EXTENDED SYNCHRONIZATION AND COMMUNICATION
FUNCTIONS

Note 1  Message is written to the message buffer area in the order of the message transmission request.

Note 2  Invoking tasks are queued to the transmission wait queue of the target message buffer in the FIFO order.

Note 3  These statements are unnecessary for the task which is created by the system configuration file because the
cfg600px generates these statement into the "kernel_id.h".

```c
#include    "kernel.h"  /*Standard header file definition*/
#include    "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1    /*Refer to note 3*/
void Task1 (VP_INT exinf);    /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ER ercd;                        /*Declares variable*/
    ID mbfid = 1;                   /*Declares and initializes variable*/
    B msg[] = {1,2,3};             /*Declares and initializes variable*/
    UINT msgsz = sizeof( msg );    /*Declares and initializes variable*/

    /* ........... */

    ercd = snd_mbf (mbfid, (VP)msg, msgsz); /*Send to message buffer*/

    if (ercd == E_OK) {
        /* ........... */ /*Normal termination processing*/
    } else if (ercd == E_RLWA1) {
        /* ........... */ /*Forced termination processing*/
    }

    /* ........... */
}
```
- **psnd_mbf, ipsnd_mbf (Polling)**

  This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

  - There is a task in the reception wait queue.
    This service call transfers the message specified by parameter `msg` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  - There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer.
    This service call stores the message specified by parameter `msg` to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

    \[
    \text{The amount of decrease = up4( msgrsz ) + VTSZ_MBFTBL}
    \]

  - There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue.
    This service call returns “E_TMOUT”.

  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ER ercd;                   /*Declares variable*/
    ID mbfid = 1;              /*Declares and initializes variable*/
    B msg[] = {1,2,3};        /*Declares and initializes variable*/
    UINT msgsz = sizeof( msg );  /*Declares and initializes variable*/

    /* .......... */
    ercd = psnd_mbf (mbfid, (VP)msg, msgsz); /*Send to message buffer*/

    if (ercd == E_OK) {
        /* .......... */
    } else if (ercd == E_TMOUT) {
        /* .......... */
    }

    /* .......... */
}
```

*Note 1* Message is written to the message buffer area in the order of the message transmission request.

*Note 2* These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
- tsnd_mbf (Wait with time-out)
   This service call processes as follows according to the situation of the message buffer specified by the parameter mbfid.

   - There is a task in the reception wait queue.
     This service call transfers the message specified by parameter msg to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

   - There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer.
     This service call stores the message specified by parameter msg to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.
     \[
     \text{The amount of decrease} = \uparrow 4(\text{msgsz}) + VTSZ_{MBFTBL}
     \]

   - There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue.
     This service call queues the invoking task to the transmission wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state with time (message transmission wait state).
     The sending WAITING state for a message buffer is cancelled in the following cases.

   The following describes an example for coding this service call.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing rcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing prcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing trcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcibly released from waiting by following either.</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- Forced release from waiting (accept ter_tsk while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- The time specified by tmout for tsnd_mbf has elapsed.</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The message buffer is reset as a result of issuing vrst_mbf.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_mbf while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1  Message is written to the message buffer area in the order of the message transmission request.

Note 2  Invoking tasks are queued to the transmission wait queue of the target message buffer in the FIFO order.

Note 3  TMO_FEVR is specified for wait time tmout, processing equivalent to snd_mbf will be executed. When TMO_POL is specified, processing equivalent to psnd_mbf will be executed.

Note 4  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

```c
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h"  /*Header file generated by cfg600px*/
#pragma task Task1  /*Refer to note 4*/
void Task1 (VP_INT exinf);  /*Refer to note 4*/
void Task1 (VP_INT exinf)
{
    ER ercd;  /*Declares variable*/
    ID mbfid = 1;  /*Declares and initializes variable*/
    B msg[] = {1,2,3};  /*Declares and initializes variable*/
    TMO tmout = 3600;  /*Declares and initializes variable*/

    /* ......... */
    ercd = tsnd_mbf (mbfid, (VP)msg, msgsz, tmout);  /*Send to message buffer*/

    if (ercd == E_OK) {
        /* ......... */  /*Normal termination processing*/
    } else if (ercd == E_RLWA1) {
        /* ......... */  /*Forced termination processing*/
    } else if (ercd == E_TMOUT) {
        /* ......... */  /*Time-out processing*/
    }
    /* ......... */
}
```
8.3.4 Receive from message buffer

A message is received (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- `rcv_mbf` (Wait)
- `prcv_mbf` (Polling)
- `trcv_mbf` (Wait with time-out)

- `rcv_mbf` (Wait)
  This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

  - There is a message in the message buffer.
    This service call takes out the oldest message from the message buffer and stores the message to the area specified by `msg` and return the size of the message. As a result, the size of available space in the target message buffer increases by the amount calculated by the following expression.
    
    \[
    \text{The amount of increase} = \text{up4( Return value )} + \text{VTSZ_MBFTBL}
    \]
    
    In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it becomes impossible to store the message in the message buffer.

    - When there is a task in the transmission wait queue and there is available space in the message buffer for the message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.
      
      \[
      \text{The amount of decrease} = \text{up4( The message size sent by the task )} + \text{VTSZ_MBFTBL}
      \]

    - There is no message in the message buffer and there is a task in the transmission wait queue.
      This service call stores the message specified by the task in the top of the transmission wait queue to the area pointed by the parameter `msg`. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
      Note, this situation is caused only when the size of the message buffer is 0.

    - There is no message in the message buffer and there is no task in the transmission wait queue.
      This service call queues the invoking task to the reception wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state (message reception wait state).

      The receiving WAITING state for a message buffer is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>snd_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>psnd_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>ipsnd_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>tsnd_mbf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbf</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1 The maximum message size is defined at creating the message buffer. The size of the area pointed by msg requires at least the maximum message size.

Note 2 Invoking tasks are queued to the reception wait queue of the target message buffer in the order of the message reception request.

Note 3 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statements into the "kernel_id.h".

```c
#include "kernel.h"  // Standard header file definition
#include "kernel_id.h"  // Header file generated by cfg600px
#pragma task
Task1  // Refer to note 3
void Task1 (VP_INT exinf);  // Refer to note 3

void Task1 (VP_INT exinf)
{
    ER ercd;  // Declares variable
    ID mbfid = 1;  // Declares and initializes variable
    B msg[16];  // Declares variable (maximum message size)

    /* ........ */
    ercd = rcv_mbf (mbfid, (VP)msg);  // Receive from message buffer

    if (ercd == E_OK) {
        /* ........ */  // Normal termination processing
    } else if (ercd == E_RLWAI) {
        /* ........ */  // Forced termination processing
    }

    /* ........ */
}
```
- **prcv_mbf** (Polling)

This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

- There is a message in the message buffer.
  This service call takes out the oldest message from the message buffer and stores the message to the area specified by `msg` and return the size of the message. As a result, the size of available space in the target message buffer increases by the amount calculated by the following expression.

  \[
  \text{The amount of increase} = \text{up4( Return value )} + \text{VTSZ_MBFTBL}
  \]

  In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it becomes impossible to store the message in the message buffer.

  - When there is a task in the transmission wait queue and there is available space in the message buffer for the message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  \[
  \text{The amount of decrease} = \text{up4( The message size sent by the task )} + \text{VTSZ_MBFTBL}
  \]

- There is no message in the message buffer and there is a task in the transmission wait queue.
  This service call stores the message specified by the task in the top of the transmission wait queue to the area pointed by the parameter `msg`. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  Note, this situation is caused only when the size of the message buffer is 0.

- There is no message in the message buffer and there is no task in the transmission wait queue.
  This service call returns “E_TMOUT”.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
Task1
/*Refer to note 2*/
void Task1 (VP_INT exinf);
/*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ER ercd;                   /*Declares variable*/
    ID mbfid = 1;              /*Declares and initializes variable*/
    B msg[16];                /*Declares variable (maximum message size)*/

    /* ............ */
    ercd = prcv_mbf (mbfid, (VP)msg); /*Receive from message buffer */

    if (ercd == E_OK) {
        /* ............ */
    } else if (ercd == E_TMOUT) {
        /* ............ */
    }

    /* ............ */
}
```

**Note 1**   The maximum message size is defined at creating the message buffer. The size of the area pointed by `msg` requires at least the maximum message size.

**Note 2**   These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”. 

- **trcv_mbf** (Wait with time-out)
  This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

  - There is a message in the message buffer.
    This service call takes out the oldest message from the message buffer and stores the message to the area specified by `msg` and return the size of the message. As a result, the size of available space in the target message buffer increases by the amount calculated by the following expression.
    \[
    \text{The amount of increase} = \text{up4( Return value )} + \text{VTSZ_MBFTBL}
    \]
    In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it becomes impossible to store the message in the message buffer.
    - When there is a task in the transmission wait queue and there is available space in the message buffer for the message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.
      \[
      \text{The amount of decrease} = \text{up4( The message size sent by the task )} + \text{VTSZ_MBFTBL}
      \]
  - There is no message in the message buffer and there is a task in the transmission wait queue.
    This service call stores the message specified by the task in the top of the transmission wait queue to the area pointed by the parameter `msg`. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
    Note, this situation is caused only when the size of the message buffer is 0.
  - There is no message in the message buffer and there is no task in the transmission wait queue.
    This service call queues the invoking task to the reception wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state with time (message reception wait state). The receiving WAITING state for a message buffer is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>snd_mbf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>psnd_mbf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>ipsnd_mbf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>tsnd_mbf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbf</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
## Chapter 8 Extended Synchronization and Communication Functions

### Note 1
The maximum message size is defined at creating the message buffer. The size of the area pointed by `msg` requires at least the maximum message size.

### Note 2
Invoking tasks are queued to the reception wait queue of the target message buffer in the order of the message reception request.

### Note 3
`TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `rcv_mbf` will be executed. When `TMO_POL` is specified, processing equivalent to `prcv_mbf` will be executed.

### Note 4
These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the “kernel_id.h”.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 4*/

void Task1 (VP_INT exinf); /*Refer to note 4*/

void Task1 (VP_INT exinf)
{
    ER ercd; /*Declares variable*/
    ID mbfid = 1; /*Declares and initializes variable*/
    B msg[16]; /*Declares variable (maximum message size)*/
    TMO tmout = 3600; /*Declares and initializes variable*/

    /* ........... */

    ercd = trcv_mbf (mbfid, (VP)msg, tmout ); /*Receive from message buffer */

    if (ercd == E_OK) {
        /* ........... */ /*Normal termination processing*/
    } else if (ercd == E_RLWA) { /*Forced termination processing*/
        /* ........... */
    } else if (ercd == E_TMOUT) { /*Time-out processing*/
        /* ........... */
    }

    /* ........... */
}
```
8.3.5 Reference message buffer state

A message buffer status is referenced by issuing the following service call from the processing program.

- ref_mbf, iref_mbf

These service calls store the detailed information of the message buffer (existence of waiting tasks, available buffer size, etc.) specified by parameter mbfid into the area specified by parameter pk_rmbf.

The following describes an example for coding this service call.

```
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#include "kernel_id.h" /*Refer to note 2*/
void Task1 (VP_INT exinf);  /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ID mbfid = 1;          /*Declares and initializes variable*/
    T_RMBF pk_rmbf;        /*Declares message structure*/
    ID stskid;             /*Declares variable*/
    ID rtskid;             /*Declares variable*/
    UINT smsgcnt;          /*Declares variable*/
    SIZE fmbfsz;           /*Declares variable*/

    /* .......... */
    ref_mbf (mbfid, &pk_rmbf);    /*Reference message buffer state*/

    stskid = pk_rmbf.stskid;     /*Acquires existence of tasks waiting for */
                                /*message transmission*/
    rtskid = pk_rmbf.rtskid;     /*Acquires existence of tasks waiting for */
                                /*message reception*/
    smsgcnt = pk_rmbf.smsgcnt;   /*Acquires the number of message in */
                                /*message buffer*/
    fmbfsz = pk_rmbf.fmbfsz;     /*Acquires the available buffer size */

    /* .......... */
}
```

Note 1 For details about the message buffer state packet, refer to "[Message buffer state packet: T_RMBF]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
CHAPTER 9 MEMORY POOL MANAGEMENT FUNCTIONS

This chapter describes the memory pool management functions performed by the RI600PX.

9.1 Outline

The RI600PX provides "Fixed-Sized Memory Pools" and "Variable-Sized Memory Pools" as dynamic memory allocation function. In the fixed-sized memory pool, the size of memory that can use is fixation, but the over-head to acquire/release memory is short. On the other hand, in the variable-sized memory pool, memory of the arbitrary size can be used, but the over-head to acquire/release memory is longer than the fixed-sized memory pool. And fragmentation of the memory pool area may occur.
9.2 Fixed-Sized Memory Pools

When a dynamic memory manipulation request is issued from a processing program in the RI600PX, the fixed-sized memory pool is provided as a usable memory area. Dynamic memory manipulation of the fixed-size memory pool is executed in fixed size memory block units.

9.2.1 Create fixed-sized memory pool

Fixed-sized memory pools are created by one of the following methods.

1) Creation by the system configuration file
   The static API "memorypool[]" described in the system configuration file creates a fixed-sized memory pool. Refer to "20.17 Fixed-sized Memory Pool Information (memorypool[])" for the details of "memorypool[]".

2) Creation by cre_mpf or acre_mpf
   The cre_mpf creates a fixed-sized memory pool with fixed-sized memory pool ID indicated by parameter mpfid according to the content of parameter pk_cmpf.
   The acre_mpf creates a fixed-sized memory pool according to the content of parameter pk_cmpf, and returns the created fixed-sized memory pool ID.
   The information specified is shown below.

   - Fixed-sized memory pool attribute (mpfatr)
     The following informations are specified as mpfatr:
     - The order of task wait queue (FIFO order or task current priority order)
   - Total number of memory blocks (blkcnt), memory block size (blksz), Start address of the fixed-sized memory pool area (mpf)
     The TSZ_MPF(blkcnt, blksz) bytes area from the address indicated by parameter mpf is used for the fixed-sized memory pool area. Refer to "18.3.3 Macros for Fixed-sized Memory Pool" for details of TSZ_MPF macro.
     The fixed-sized memory pool area should be in the memory object to be able to access by tasks which uses memory blocks.
   - Start address of the fixed-sized memory pool management area (mpfmb)
     The TSZ_MPMB(blkcnt, blksz) bytes area from the address indicated by parameter mpfmb is used for the fixed-sized memory pool management area. Refer to "18.3.3 Macros for Fixed-sized Memory Pool" for details of TVSZ_MPMB macro.
     The fixed-sized memory pool management area should be generated to the area other than memory objects and user stacks.

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_mpf as a representative.
Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#define BLKCNT    32                /*Total number of memory blocks*/
#define BLKSZ     16                /*Memory block size (in bytes)*/
#pragma section B BU_SH           /*Section for the fixed-sized memory pool area*/
    /*Fixed-sized memory pool area*/
static UW mpf_area[ TSZ_MPF(BLKCNT, BLKSZ)/sizeof(UW) ];
    /* Section for the fixed-sized memory pool management area*/
#pragma section B BRI_RAM
    /*Fixed-sized memory pool management area*/
static UW mpfmb_area[ TSZ_MPFMB(BLKCNT, BLKSZ)/sizeof(UW) ];
#pragma section
#pragma task Task1
    /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER mpfid;        /*Declares variable*/
    T_CMPF pk_cmpf = { /*Declares and initializes variable*/
        TA_TFIFO,       /*Fixed-sized memory pool attribute (mpfatr)*/
        BLKCNT,         /*Total number of memory blocks (blkcnt)*/
        BLKSZ,          /*Memory block size (in bytes) (blksz)*/
        (VP)mpf_area,    /*Start address of the fixed-sized memory pool area (mpf)*/
        (VP)mpfmb_area   /*Start address of the fixed-sized memory pool management area (mpfmb)*/
    };
    /* ........ */
    mpfid = acre_mpf ( &pk_cmpf );  /*Create fixed-sized memory pool/
    /* ........ */
}
```
9.2.2 Delete fixed-sized memory pool

- del_mpf
  This service call deletes the fixed-sized memory pool specified by parameter mpfid.
  When there are waiting tasks for the target fixed-sized memory pool by using get_mpf or tget_mpf, this service call
  cancels the WAITING state of the tasks and returns E_DLT as a return value of the get_mpf or tget_mpf.
  This service call can be called from tasks that belong to Trusted Domain.
  The following describes an example for coding this service call.

```
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/

void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ID mpfid = 8;          /*Declares and initializes variable*/

    /* .......... */

    ercd = del_mpf ( mpfid );   /*Delete fixed-sized memory pool*/

    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the
cfg600px generates these statement into the “kernel_id.h”.

9.2.3 Acquire fixed-sized memory block

A fixed-sized memory block is acquired (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- `get_mpf (Wait)`
- `pget_mpf, ipget_mpf (Polling)`
- `tget_mpf (Wait with time-out)`

The RI600PX does not perform memory clear processing when a fixed-sized memory block is acquired. The contents of the acquired fixed-size memory block are therefore undefined.

- `get_mpf (Wait)`

  This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter `mpfid` and stores the start address in the area specified by parameter `p_blk`.

  If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (no available fixed-size memory blocks exist) when this service call is issued, this service call does not acquire the fixed-size memory block but queues the invoking task to the target fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state (fixed-size memory block acquisition wait state).

  The WAITING state for a fixed-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Fixed-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>rel_mpf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>irel_mpf</code>.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>The fixed-sized memory pool is reset as a result of issuing <code>vrst_mpf</code>.</td>
<td><code>EV_RST</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mpf</code> while waiting).</td>
<td><code>E_DLT</code></td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note 4*/
void Task1 (VP_INT exinf);          /*Refer to note 4*/

void Task1 (VP_INT exinf) {
    ER ercd;                   /*Declares variable*/
    ID mpfid = 1;              /*Declares and initializes variable*/
    VP p_blk;                  /*Declares variable*/

    /* ........ * /
    ercd = get_mpf (mpfid, &p_blk); /*Acquire fixed-sized memory block */
    if (ercd == E_OK) {
        /* ........ */
        rel_mpf (mpfid, p_blk);    /*Release fixed-sized memory block*/
    } else if (ercd == E_RLWAI) {
        /* ........ */
    } else if (ercd == EV_RST) {
    } else if (ercd == E_DLT) {
    }

    /* ........ */
}
```
Note 1  Invoking tasks are queued to the target fixed-size memory pool wait queue in the order defined at creating the fixed-sized memory pool (FIFO order or current priority order).

Note 2  The contents of the block are undefined.

Note 3  The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.

- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

Note 4  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- **pget_mpf, ipget_mpf** (Polling)

  This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter `mpfid` and stores the start address in the area specified by parameter `p_blk`.

  If a fixed-sized memory block could not be acquired from the target fixed-sized memory pool (no available fixed-sized memory blocks exist) when this service call is issued, fixed-sized memory block acquisition processing is not performed but “E_TMOUT” is returned.

  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                 /*Refer to note 3*/
void Task1 (VP_INT exinf);          /*Refer to note 3*/

void Task1 (VP_INT exinf) {            
  ER ercd;                               /*Declares variable*/
  ID mpfid = 1;                          /*Declares and initializes variable*/
  VP p_blk;                              /*Declares variable*/

  /* ............ */                      /*Acquire fixed-sized memory block */

  ercd = pget_mpf (mpfid, &p_blk);

  if (ercd == E_OK) {                    /*Polling success processing*/
    /* ............ */

    rel_mpf (mpfid, p_blk);              /*Release fixed-sized memory block*/
  } else if (ercd == E_TMOUT) {         /*Polling failure processing*/
    /* ............ */
  }

  /* ............ */
}
```

**Note 1** The contents of the block are undefined.

**Note 2** The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

**Note 3** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
- **tget_mpf (Wait with time-out)**
  This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter `mpfid` and stores the start address in the area specified by parameter `p_blk`.
  If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (no available fixed-size memory blocks exist) when this service call is issued, this service call does not acquire the fixed-size memory block but queues the invoking task to the target fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state with time-out (fixed-size memory block acquisition wait state).
  The WAITING state for a fixed-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Fixed-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>rel_mpf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>irel_mpf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The fixed-sized memory pool is reset as a result of issuing <code>vrst_mpf</code>.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mpf</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.

```
#include "kernel.h"            /*Standard header file definition*/
#include "kernel_id.h"          /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 5*/
void Task1 (VP_INT exinf);     /*Refer to note 5*/

void Task1 (VP_INT exinf)
{
    ER ercd;                      /*Declares variable*/
    ID mpfid = 1;                 /*Declares and initializes variable*/
    VP p_blk;                     /*Declares variable*/
    TMO tmout = 3600;             /*Declares and initializes variable*/

    /* .......... */               /*Acquire fixed-sized memory block*/
    ercd = tget_mpf (mpfid, &p_blk, tmout);

    if (ercd == E_OK) {
        /* .......... */            /*Normal termination processing*/
        rel_mpf (mpfid, p_blk);   /*Release fixed-sized memory block*/
    } else if (ercd == E_RLWAI) {
        /* .......... */            /*Forced termination processing*/
    } else if (ercd == E_TMOUT) {
        /* .......... */            /*Time-out processing*/
    }

    /* .......... */
}
```

**Note 1**  Invoking tasks are queued to the target fixed-size memory pool wait queue in the order defined at creating the fixed-sized memory pool (FIFO order or current priority order).

**Note 2**  The contents of the block are undefined.
Note 3 The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.
- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

Note 4 TMO_FEVR is specified for wait time \( \text{tmout} \), processing equivalent to get\_mpf will be executed. When TMO\_POL is specified, processing equivalent to pget\_mpf will be executed.

Note 5 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel\_id.h”.
9.2.4 Release fixed-sized memory block

A fixed-sized memory block is returned by issuing the following service call from the processing program.

- `rel_mpf, irel_mpf`

  This service call returns the fixed-sized memory block specified by parameter `blk` to the fixed-sized memory pool specified by parameter `mpfid`.

  If a task is queued to the target fixed-sized memory pool wait queue when this service call is issued, fixed-sized memory block return processing is not performed but fixed-sized memory blocks are returned to the relevant task (first task of wait queue).

  As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (WAITING state for a fixed-sized memory block) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ER ercd;                   /*Declares variable*/
    ID mpfid = 1;              /*Declares and initializes variable*/
    VP blk;                    /*Declares variable*/

    /* ......... */
    ercd = get_mpf (mpfid, &blk);   /*Acquire fixed-sized memory block */
    /*(waiting forever)*/

    if (ercd == E_OK) {
        /* ......... */
    /*Normal termination processing*/

        rel_mpf (mpfid, blk);       /*Release fixed-sized memory block*/
    } else if (ercd == E_RLWAI) {
        /* ......... */
    /*Forced termination processing*/
    }

    /* ......... */
}
```

**Note** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
9.2.5 Reference fixed-sized memory pool state

A fixed-sized memory pool status is referenced by issuing the following service call from the processing program.

- `ref_mpf`, `iref_mpf`
  Stores fixed-sized memory pool state packet (ID number of the task at the head of the wait queue, number of free memory blocks, etc.) of the fixed-sized memory pool specified by parameter `mpfid` in the area specified by parameter `pk_rmpf`.
  The following describes an example for coding these service calls.

```c
#include "kernel.h"          /*Standard header file definition*/
#include "kernel_id.h"        /*Header file generated by cfg600px*/

#pragma task Task1
void Task1 (VP_INT exinf);  /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ID mpfid = 1;              /*Declares and initializes variable*/
    T_RMPF pk_rmpf;            /*Declares data structure*/
    ID wtskid;                /*Declares variable*/
    UINT fblkcnt;             /*Declares variable*/

    /* .......... */

    ref_mpf (mpfid, &pk_rmpf);   /*Reference fixed-sized memory pool state*/
    wtskid = pk_rmpf.wtskid;    /*Reference ID number of the task at the */
                                /*head of the wait queue*/
    fblkcnt = pk_rmpf.fblkcnt;  /*Reference number of free memory blocks*/

    /* .......... */
}
```

Note 1 For details about the fixed-sized memory pool state packet, refer to "[Fixed-sized memory pool state packet: T_RMPF]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
9.3 Variable-Sized Memory Pools

When a dynamic memory manipulation request is issued from a processing program in the RI600PX, the variable-sized memory pool is provided as a usable memory area. Dynamic memory manipulation for variable-size memory pools is performed in the units of the specified variable-size memory block size.

9.3.1 Size of Variable-sized memory block

In the current implementation of the RI600PX, the size of the variable-sized memory block to be acquired is selected from 12 (in maximum) kinds of variations. This variations are selected from 24 kinds of inside decided beforehand according to the maximum memory block size that is defined at creating the variable-sized memory pool. Table 9-1 shows variation of memory block size. Note, this behavior may be changed in the future version.

<table>
<thead>
<tr>
<th>No,</th>
<th>Size of memory block (Hexadecimal)</th>
<th>Example-1 max_memsize = 0x100</th>
<th>Example-1 max_memsize = 0x2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12 (0xC)</td>
<td>Used</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>36 (0x24)</td>
<td>Used</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>84 (0x54)</td>
<td>Used</td>
<td>Used</td>
</tr>
<tr>
<td>4</td>
<td>180 (0xB4)</td>
<td>Used</td>
<td>Used</td>
</tr>
<tr>
<td>5</td>
<td>372 (0x174)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>6</td>
<td>756 (0x2F4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>7</td>
<td>1524 (0x5F4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>8</td>
<td>3060 (0xBF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>9</td>
<td>6132 (0x17F4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>10</td>
<td>12276 (0x2FF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>11</td>
<td>24564 (0x5FF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>12</td>
<td>49140 (0xBFF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>13</td>
<td>98292 (0x17FF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>14</td>
<td>196596 (0x2FFFF4)</td>
<td>-</td>
<td>Used</td>
</tr>
<tr>
<td>15</td>
<td>393204 (0x5FFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>786420 (0xBFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>1572852 (0x17FFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>3145716 (0x2FFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>6291444 (0x5FFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>12582900 (0xBFFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>21</td>
<td>25165812 (0x17FFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>22</td>
<td>50331636 (0x2FFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>23</td>
<td>100663284 (0x5FFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td>201326580 (0xBFFFFF4)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
9.3.2 Create variable-sized memory pool

Variable-sized memory pools are created by one of the following methods.

1) Creation by the system configuration file
   The static API "variable_memorypool[]" described in the system configuration file creates a variable-sized memory pool.
   Refer to "20.18 Variable-sized Memory Pool Information (variable_memorypool[])" for the details of "variable_memorypool[]".

2) Creation by cre_mpl or acre_mpl
   The cre_mpl creates a variable-sized memory pool with variable-sized memory pool ID indicated by parameter mplid according to the content of parameter pk_cmpl.
   The acre_mpl creates a variable-sized memory pool according to the content of parameter pk_cmpl, and returns the created variable-sized memory pool ID.
   The information specified is shown below.
   - Variable-sized memory pool attribute (mplatr)
     Only TA_TFIFO (the order of task wait queue is managed by FIFO order.) can be specified for mplatr.
   - Size of variable-sized memory pool area (mplsz), Start address of the variable-sized memory pool area (mpl)
     The start addresess of the variable-sized memory pool area must be 4-bytes boundary.
     The variable-sized memory pool area should be in the memory object to be able to access by tasks which uses memory blocks.
   - Maximum memory block size (maxblksz)
     Specify the maximum memory block size for maxblksz. For details, refer to "9.3.1 Size of Variable-sized memory block".

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_mpl as a representative.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#define MPLSZ     1024      /*Size of variable-sized memory pool area (in bytes)*/
#define MAXBLKLSZ 128       /*Maximum memory block size (in bytes)*/

#pragma section B BU_SH     /*Section for the variable-sized memory pool area*/
static UW mpl_area[ MPLSZ/sizeof(UW) ]; /*Variable-sized memory pool area*/
#pragma section
#pragma task

void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER mplid;        /*Declares variable*/
    T_CMPF pk_cmpl = { /*Declares and initializes variable*/
        TA_TFIFO,  /*Variable-sized memory pool attribute (mplatr)*/
        MPLSZ,    /*Size of variable-sized memory pool area (in bytes) (mplsz)*/
        (VP)mpl_area,/*Start address of the variable-sized memory pool area (mpl)*/
        MAXBLKLSZ /*Maximum memory block size (in bytes) (maxblksz)*/
    };

    /* ............ */
    mplid = acre_mpl ( &pk_cmpl ); /*Create variable-sized memory pool/
    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
9.3.3 Delete variable-sized memory pool

- del_mpl

This service call deletes the variable-sized memory pool specified by parameter mplid.

When there are waiting tasks for the target variable-sized memory pool by using get_mpl or tget_mpl, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the get_mpl or tget_mpl.

This service call can be called from tasks that belong to Trusted Domain.

The following describes an example for coding this service call.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note*/
void Task1 (VP_INT exinf); /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID mplid = 8; /*Declares and initializes variable*/

    /* ........ */

ercd = del_mpl ( mplid ); /*Delete variable-sized memory pool*/

    /* ........ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
### 9.3.4 Acquire variable-sized memory block

A variable-sized memory block is acquired (waiting forever, polling, or with time-out) by issuing the following service call from the processing program.

- `get_mpl` (Wait)
- `pget_mpl`, `ipget_mpl` (Polling)
- `tget_mpl` (Wait with time-out)

The RI600PX does not perform memory clear processing when a variable-sized memory block is acquired. The contents of the acquired variable-size memory block are therefore undefined.

- `get_mpl` (Wait)
  
  This service call acquires a variable-size memory block of the size specified by parameter `blksz` from the variable-size memory pool specified by parameter `mplid`, and stores its start address into the area specified by parameter `p_blk`.

  If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory blocks but queues the invoking task to the target variable-size memory pool wait queue and moves it from the RUNNING state to the WAITING state (variable-size memory block acquisition wait state).

  The WAITING state for a variable-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Variable-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable-size memory block that satisfies the requested size was returned to the target variable-size memory pool as a result of issuing rel_mpl.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcibly released from waiting by following either.</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- Forced release from waiting (accept ter_tsk while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- The time specified by <code>tmout</code> for <code>tget_mpl</code> has elapsed.</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The variable-sized memory pool is reset as a result of issuing <code>vrst_mpl</code>.</td>
<td>E_V_RST</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mpl</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following describes an example for coding this service call.
Note 1 For the size of the memory block, refer to "9.3.1 Size of Variable-sized memory block".

Note 2 Invoking tasks are queued to the target variable-size memory pool wait queue in the FIFO order.

Note 3 The contents of the block are undefined.

Note 4 The alignment number of memory blocks changes with creation method of the variable-sized memory pool.

- Created by system configuration file
  The alignment number is 1. To enlarge the alignment number to 4, specify unique section to Section name assigned to the memory pool area (mpl_section) in Variable-sized Memory Pool Information (variable_memorypool[]) and locate the section to 4-bytes boundary address when linking.

- Created by cre_mpl or acre_mpl
  The alignment number is 4.

Note 5 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include    "kernel.h"    /*Standard header file definition*/
#include    "kernel_id.h"   /*Header file generated by cfg600px*/
#pragma task Task1    /*Refer to note 5*/
void Task1 (VP_INT exinf);    /*Refer to note 5*/
void Task1 (VP_INT exinf) {
   ER  ercd;    /*Declares variable*/
   ID  mplid = 1;    /*Declares and initializes variable*/
   UINT  blksz = 256;    /*Declares and initializes variable*/
   VP  p_blk;    /*Declares variable*/
   /* ......... */ /*Acquire variable-sized memory block */
   ercd = get_mpl (mplid, blksz, &p_blk);
   if (ercd == E_OK) {
      /* ......... */ /*Normal termination processing*/
      rel_mpl (mplid, p_blk); /*Release variable-sized memory block*/
   } else if (ercd == E_RLWA1) {
      /* ......... */ /*Forced termination processing*/
   }
   /* ......... */
}
```
- pget_mpl, ipget_mpl (Polling)
  This service call acquires a variable-size memory block of the size specified by parameter \texttt{blksz} from the variable-size memory pool specified by parameter \texttt{mplid}, and stores its start address into the area specified by parameter \texttt{p_blk}.

  If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory block but returns "E_TMOUT".

  The following describes an example for coding these service calls.

  \begin{verbatim}
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                 /*Refer to note 4*/
void TaskI (VP_INT exinf);         /*Refer to note 4*/

void TaskI (VP_INT exinf)
{
  ER ercd;                   /*Declares variable*/
  ID mplid = 1;              /*Declares and initializes variable*/
  UINT blksz = 256;          /*Declares and initializes variable*/
  VP   p_blk;                /*Declares variable*/

  /* .......... */  /*Acquire variable-sized memory block*/
  ercd = pget_mpl (mplid, blksz, &p_blk);

  if (ercd == E_OK) {
    /* .......... */  /*Polling success processing*/
      rel_mpl (mplid, p_blk);  /*Release variable-sized memory block*/
  } else if (ercd == E_TMOUT) {
    /* .......... */  /*Polling failure processing*/
  }

  /* .......... */
}
\end{verbatim}

Note 1  For the size of the memory block, refer to "9.3.1 Size of Variable-sized memory block".

Note 2  The contents of the block are undefined.

Note 3  The alignment number of memory blocks changes with creation method of the variable-sized memory pool.

  - Created by system configuration file
    The alignment number is 1. To enlarge the alignment number to 4, specify unique section to \texttt{Section name assigned to the memory pool area (mpl_section)} in \texttt{Variable-sized Memory Pool Information (variable_memorypool[]) and locate the section to 4-bytes boundary address when linking.}

  - Created by \texttt{cre_mpl} or \texttt{acre_mpl}
    The alignment number is 4.

Note 4  These statements are unnecessary for the task which is created by the system configuration file because the \texttt{cfg600px} generates these statement into the "kernel_id.h".
- `tget_mpl` (Wait with time-out)
  
  This service call acquires a variable-size memory block of the size specified by parameter `blksz` from the variable-size memory pool specified by parameter `mplid`, and stores its start address into the area specified by parameter `p_blk`. If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory blocks but queues the invoking task to the target variable-size memory pool wait queue and moves it from the RUNNING state to the WAITING state with time-out (variable-size memory block acquisition wait state).

  The WAITING state for a variable-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Variable-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable-size memory block that satisfies the requested size was returned to the target variable-size memory pool as a result of issuing <code>rel_mpl</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcibly released from waiting by following either.</td>
<td></td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>- Forced release from waiting (accept <code>ter_tsk</code> while waiting).</td>
<td></td>
</tr>
<tr>
<td>- The time specified by <code>tmout</code> for <code>tget_mpl</code> has elapsed.</td>
<td></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The variable-sized memory pool is reset as a result of issuing <code>vrst_mpl</code>.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mpl</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

  The following describes an example for coding this service call.
Note 1 For the size of the memory block, refer to "9.3.1 Size of Variable-sized memory block".

Note 2 Invoking tasks are queued to the target variable-size memory pool wait queue in the FIFO order.

Note 3 The contents of the block are undefined.

Note 4 The alignment number of memory blocks changes with creation method of the variable-sized memory pool.
- Created by system configuration file
  The alignment number is 1. To enlarge the alignment number to 4, specify unique section to Section name assigned to the memory pool area (mpl_section) in Variable-sized Memory Pool Information (variable_memorypool[]) and locate the section to 4-bytes boundary address when linking.
- Created by cre_mpl or acre_mpl
  The alignment number is 4.

Note 5 TMO_FEVR is specified for wait time timeout, processing equivalent to get_mpl will be executed. When TMO_POL is specified, processing equivalent to pget_mpl will be executed.

Note 6 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task
Task1 /*Refer to note 4*/
void Task1 (VP_INT exinf)
{
  ER ercd; /*Declares variable*/
  ID mplid = 1; /*Declares and initializes variable*/
  UINT blksz = 256; /*Declares and initializes variable*/
  VP p_blk; /*Declares variable*/
  TMO tmout = 3600; /*Declares and initializes variable*/

  /* ........ */

  /*Acquire variable-sized memory block*/
  ercd = tget_mpl (mplid, blksz, &p_blk, tmout);
  if (ercd == E_OK) {
    /* ........ */ /*Normal termination processing*/
    rel_mpl (mplid, p_blk); /*Release variable-sized memory block*/
  } else if (ercd == E_RLWA1) {
    /* ........ */ /*Forced termination processing*/
  } else if (ercd == E_TMOUT) {
    /* ........ */ /*Time-out processing*/
  }

  /* ........ */
}
```
9.3.5 Release variable-sized memory block

A variable-sized memory block is returned by issuing the following service call from the processing program.

- `rel_mpl`

This service call returns the variable-sized memory block specified by parameter `blk` to the variable-sized memory pool specified by parameter `mplid`.

After returning the variable-size memory blocks, these service calls check the tasks queued to the target variable-size memory pool wait queue from the top, and assigns the memory if the size of memory requested by the wait queue is available. This operation continues until no tasks queued to the wait queue remain or no memory space is available. As a result, the task that acquired the memory is unlinked from the queue and moved from the WAITING state (variable-size memory block acquisition wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

The following describes an example for coding these service calls.

```c
#include "kernel.h"          /*Standard header file definition*/
#include "kernel_id.h"       /*Header file generated by cfg600px*/
#pragma task Task1            /*Refer to note 2*/

void Task1 (VP_INT exinf); /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;                  /*Declares variable*/
    ID mplid = 1;             /*Declares and initializes variable*/
    UINT blksz = 256;        /*Declares and initializes variable*/
    VP blk;                  /*Declares variable*/

    /* ......... */          /*Acquire variable-sized memory block*/
    ercd = get_mpl (mplid, blksz, &blk);

    if (ercd == E_OK) {
        /* ......... */     /*Normal termination processing*/
        rel_mpl (mplid, blk);  /*Release variable-sized memory block*/
    } else if (ercd == E_RLWAI) {
        /* ......... */      /*Forced termination processing*/
    }

    /* ......... */
}
```

Note 1 The RI600PX do only simple error detection for `blk`. If `blk` is illegal and the error is not detected, the operation is not guaranteed after that.

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".

```c
#include "kernel.h"          /*Standard header file definition*/
#include "kernel_id.h"       /*Header file generated by cfg600px*/
#pragma task Task1            /*Refer to note 2*/

void Task1 (VP_INT exinf); /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER ercd;                  /*Declares variable*/
    ID mplid = 1;             /*Declares and initializes variable*/
    UINT blksz = 256;        /*Declares and initializes variable*/
    VP blk;                  /*Declares variable*/

    /* ......... */          /*Acquire variable-sized memory block*/
    ercd = get_mpl (mplid, blksz, &blk);

    if (ercd == E_OK) {
        /* ......... */     /*Normal termination processing*/
        rel_mpl (mplid, blk);  /*Release variable-sized memory block*/
    } else if (ercd == E_RLWAI) {
        /* ......... */      /*Forced termination processing*/
    }

    /* ......... */
}
```
9.3.6 Reference variable-sized memory pool state

A variable-sized memory pool status is referenced by issuing the following service call from the processing program.

- ref_mpl, iref_mpl

These service calls store the detailed information (ID number of the task at the head of the wait queue, total size of free memory blocks, etc.) of the variable-size memory pool specified by parameter mplid into the area specified by parameter pk Ramirez.

The following describes an example for coding these service calls.

```c
#include "kernel.h"     /*Standard header file definition*/
#include "kernel_id.h"   /*Header file generated by cfg600px*/
#include <stdio.h>        /*Standard library file definition*/

#define RAMPL "T_RMPL"    /*Define a macro for variable-sized memory pool state packet*/
#define TASK Name "Task1" /*Define a macro for task name*/

void Task1 (VP_INT exinf);    /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ID mplid = 1;              /*Declares and initializes variable*/
    T_RMPL pk_rmpl;            /*Declares data structure*/
    ID wtskid;                 /*Declares variable*/
    SIZE fmplesz;              /*Declares variable*/
    UINT fblksz;               /*Declares variable*/
    /* ......... */
    ref_mpl (mplid, &pk_rmpl); /*Reference variable-sized memory pool state*/
    wtskid = pk_rmpl.wtskid;   /*Reference ID number of the task at the */
                               /*head of the wait queue*/
    fmplesz = pk_rmpl.fmplesz; /*Reference total size of free memory blocks*/
    fblksz = pk_rmpl.fblksz;   /*Reference maximum memory block size*/
    /* ......... */
}
```

Note 1 For details about the variable-sized memory pool state packet, refer to "[Variable-sized memory pool state packet: T_RMPL]."

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
CHAPTER 10 TIME MANAGEMENT FUNCTIONS

This chapter describes the time management functions performed by the RI600PX.

10.1 Outline

The RI600PX’s time management function provides methods to implement time-related processing (Timer Operations: Delay Task, Time-out, Cyclic Handlers, Alarm Handlers and System Time) by using base clock timer interrupts that occur at constant intervals, as well as a function to manipulate and reference the system time.

10.2 System Time

The system time is a time used by the RI600PX for performing time management (in millisecond). After initialization to 0 by the Kernel Initialization Module (vsta_knl, ivsta_knl), the system time is updated based on the base clock interval defined by Denominator of base clock interval time (tic_deno) and Denominator of base clock interval time (tic_deno) in System Information (system) when creating a system configuration file.

10.2.1 Base clock timer interrupt

To realize the time management function, the RI600PX uses interrupts that occur at constant intervals (base clock timer interrupts). When a base clock timer interrupt occurs, processing related to the RI600PX time (system time update, task time-out/delay, cyclic handler activation, alarm handler activation, etc.) is executed. Basically, either of channel 0-3 of the compare match timer (CMT) implemented in the MCU is used for base clock time. The channel number is specified by Selection of timer channel for base clock (timer) in Base Clock Interrupt Information (clock) in the system configuration file. The hardware initialization to generate base clock timer interrupt is separated as user-own coding module. For details, refer to “10.9 Base Clock Timer Initialization Routine (_RI_init_cmt_knl( ))”

10.2.2 Base clock interval

In the RI600PX, service call parameters for time specification are specified in msec units. It is desirable to set 1 msec for the occurrence interval of base clock timer interrupts, but it may be difficult depending on the target system performance (processing capability, required time resolution, or the like). In such a case, the occurrence interval of base clock timer interrupt can be specified by Denominator of base clock interval time (tic_deno) and Denominator of base clock interval time (tic_deno) in System Information (system) when creating a system configuration file. By specifying the base clock interval, processing regards that the time equivalent to the base clock interval elapses during a base clock timer interrupt.
10.3 Timer Operations

The RI600PX’s timer operation function provides Delay Task, Time-out, Cyclic Handlers, Alarm Handlers and System Time, as the method for realizing time-dependent processing.

10.4 Delay Task

Delayed task that makes the invoking task transit from the RUNNING state to the WAITING state during the interval until a given length of time has elapsed, and makes that task move from the WAITING state to the READY state once the given length of time has elapsed.

Delayed wake-up is implemented by issuing the following service call from the processing program.

\[ \text{dly_tsk} \]

10.5 Time-out

Time-out is the operation that makes the target task move from the RUNNING state to the WAITING state during the interval until a given length of time has elapsed if the required condition issued from a task is not immediately satisfied, and makes that task move from the WAITING state to the READY state regardless of whether the required condition is satisfied once the given length of time has elapsed.

A time-out is implemented by issuing the following service call from the processing program.

\[ \text{tslp_tsk, twai_sem, twai_flg, tsnd_dtq, trcv_dtq, trcv_mbx, tloc_mtx, tsnd_mbf, trcv_mbf, tget_mpf, tget_mpl} \]
10.6 Cyclic Handlers

The cyclic handler is a routine dedicated to cycle processing that is activated periodically at a constant interval (activation cycle). The RI600PX handles the cyclic handler as a “non-task (module independent from tasks)”. Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when a specified activation cycle has come, and the control is passed to the cyclic handler.

10.6.1 Basic form of cyclic handler

The following shows the basic form of cyclic handlers. The extended information defined at creating the cyclic handler is passed to the `exinf`

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma cychandler Cychdr1 /*Refer to note*/

void Cychdr1 (VP_INT exinf);

void Cychdr1 (VP_INT exinf)
{
    /* ......... */
    return;                         /*Terminate cyclic handler*/
}
```

Note These statements are unnecessary for the cyclic handler which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

10.6.2 Processing in cyclic handler

- Stack
  A cyclic handler uses the system stack.

- Service call
  The RI600PX handles the cyclic handler as a "non-task".
  The cyclic handler can issue service calls whose "Useful range" is "Non-task".

Note If a service call (isig_sem, iset_flg, etc.) which causes dispatch processing (task scheduling processing) is issued in order to quickly complete the processing in the cyclic handler during the interval until the processing in the cyclic handler ends, the RI600PX executes only processing such as queue manipulation, counter manipulation, etc., and the actual dispatch processing is delayed until a return instruction is issued by the cyclic handler, upon which the actual dispatch processing is performed in batch.

- PSW register when processing is started

Table 10-1 PSW Register When Cyclic Handler is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td>Base clock interrupt priority level (IPL)</td>
<td>Do not lower IPL more than the start of process-</td>
</tr>
<tr>
<td>PM</td>
<td>0</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>System stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
10.6.3 Create cyclic handler

Cyclic handlers are created by one of the following methods.

1) Creation by the system configuration file

The static API “cyclic_hand[]” described in the system configuration file creates a cyclic handler.
Refer to “20.19 Cyclic Handler Information (cyclic_hand[])” for the details of “cyclic_hand[]”.

2) Creation by cre_cyc or acre_cyc

The cre_cyc creates a cyclic handler with cyclic handler ID indicated by parameter cycid according to the content of parameter pk_ccyc.
The acre_cyc creates a cyclic handler according to the content of parameter pk_ccyc, and returns the created cyclic handler ID.
The information specified is shown below.

- Cyclic handler attribute (cycatr)
The following informations are specified as cycatr:
  - Whether the cyclic handler is in the operational state. (TA_STA attribute)
  - Whether the activation phase is saved. (TA_PHS attribute)

- Extended information (exinf)

- Cyclic handler start address (cychdr)
- Activation cycle (cyctim)
- Activation phase (cycphs)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_cyc as a representative.

```c
#include    "kernel.h"     /*Standard header file definition*/
#include    "kernel_id.h"     /*Header file generated by cfg600px*/

extern void Cychdr1(VP_INT exinf);

#pragma task Task1       /*Refer to note*/
void Task1 (VP_INT exinf);     /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER cycid;                     /*Declares variable*/
    T_CCYC pk_ccyc = {         /*Declares and initializes variable*/
        TA_STA,                /*Cyclic handler attribute (cycatr)*/
        0,                     /*Extended information (exinf)*/
        (FP) Cychdr1,         /*Start address (cychdr)*/
        10,                    /*Activation cycle (cyctim)*/
        2                      /*Activation phase (cycphs)*/
    };

    /* ............ */

    cycid = acre_cyc ( &pk_ccyc );     /*Create cyclic handler*/

    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
10.6.4 Delete cyclic handler

- del_cyc

This service call deletes the cyclic handler specified by parameter cycid.
This service call can be called from tasks that belong to Trusted Domain.
The following describes an example for coding this service call.

```
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ID cycid = 8;             /*Declares and initializes variable*/
    /* ............ */
    ercd = del_cyc (cycid);  /*Delete cyclic handler*/
    /* ............ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
### 10.6.5 Start cyclic handler operation

Moving to the operational state (STA state) is implemented by issuing the following service call from the processing program.

- **sta_cyc, ista_cyc**
  
  This service call moves the cyclic handler specified by parameter `cycid` from the non-operational state (STP state) to operational state (STA state).

As a result, the target cyclic handler is handled as an activation target of the RI600PX.

The relative interval from when either of this service call is issued until the first activation request is issued varies depending on whether the **TA_PHS attribute (phsatr)** is specified for the target cyclic handler during configuration.

- If the TA_PHS attribute is specified
  
  The target cyclic handler activation timing is set up based on the activation phases and activation cycle. If the target cyclic handler has already been in operational state, however, no processing is performed even if this service call is issued, but it is not handled as an error.

  The following shows a cyclic handler activation timing image.

  ![Figure 10-1 TA_PHS Attribute: Specified](image)

- If the TA_PHS attribute is not specified
  
  The target cyclic handler activation timing is set up according to the activation cycle on the basis of the call time of this service call.

  This setting is performed regardless of the operating status of the target cyclic handler.

  The following shows a cyclic handler activation timing image.

  ![Figure 10-2 TA_PHS Attribute: Not Specified](image)
The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1              /*Refer to note*/
void Task1 (VP_INT exinf);      /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ID cycid = 1;                  /*Declares and initializes variable*/
    /* ........ */
    sta_cyc (cycid);              /*Start cyclic handler operation*/
    /* ........ */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
10.6.6  Stop cyclic handler operation

Moving to the non-operational state (STP state) is implemented by issuing the following service call from the processing program.

- **stp_cyc, istp_cyc**

  This service call moves the cyclic handler specified by parameter `cycid` from the operational state (STA state) to non-operational state (STP state).

  As a result, the target cyclic handler is excluded from activation targets of the RI600PX until issuance of `sta_cyc` or `ista_cyc`.

  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID cycid = 1;              /* Declares and initializes variable*/
    /* ......... */
    stp_cyc (cycid);                /*Stop cyclic handler operation*/
    /* ......... */
}
```

**Note 1**  This service call does not perform queuing of stop requests. If the target cyclic handler has been moved to the non-operational state (STP state), therefore, no processing is performed but it is not handled as an error.

**Note 2**  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.6.7 Reference cyclic handler state

A cyclic handler status by issuing the following service call from the processing program.

- `ref_cyc`, `iref_cyc`

Stores cyclic handler state packet (current state, time until the next activation, etc.) of the cyclic handler specified by parameter `cycid` in the area specified by parameter `pk_rcyc`. The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1
/*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ID     cycid = 1;              /*Declares and initializes variable*/
    T_RCYC pk_rcyc;                /*Declares data structure*/
    STAT   cycstat;                /*Declares variable*/
    RELTIM lefttim;                /*Declares variable*/
    /* ......... */
    ref_cyc (cycid, &pk_rcyc);     /*Reference cyclic handler state*/
    cycstat = pk_rcyc.cycstat;     /*Reference current state*/
    lefttim = pk_rcyc.lefttim;     /*Reference time left before the next */
                                    /*activation*/
    /* ......... */
}
```

Note 1 For details about the cyclic handler state packet, refer to "[Cyclic handler state packet: T_RCYC]."

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.7 Alarm Handlers

The alarm handler is a routine started when the specified time passes. The RI600PX handles the alarm handler as a "non-task (module independent from tasks)". Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when a specified time has elapsed, and the control is passed to the alarm handler.

10.7.1 Basic form of alarm handler

The following shows the basic form of alarm handlers. The extended information defined at creating the alarm handler is passed to the exinf.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma alarmhand Almhdr1           /*Refer to note*/
void       Almhdr1 (VP_INT exinf); /*Refer to note*/

void       Almhdr1 (VP_INT exinf)
{
    /* .......... */

    return;                                    /*Terminate alarm handler*/
}
```

Note These statements are unnecessary for the alarm handler which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

10.7.2 Processing in alarm handler

- Stack
  A alarm handler uses the system stack.

- Service call
  The RI600PX handles the alarm handler as a “non-task”.
  The alarm handler can issue service calls whose “Useful range” is “Non-task”.

  Note  If a service call (isig_sem, iset_flg, etc.) which causes dispatch processing (task scheduling processing) is issued in order to quickly complete the processing in the alarm handler during the interval until the processing in the alarm handler ends, the RI600PX executes only processing such as queue manipulation, counter manipulation, etc., and the actual dispatch processing is delayed until a return instruction is issued by the alarm handler, upon which the actual dispatch processing is performed in batch.

- PSW register when processing is started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td>Base clock interrupt priority level (IPL)</td>
<td>Do not lower IPL more than the start of processing.</td>
</tr>
<tr>
<td>PM</td>
<td>0</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>System stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
10.7.3 Create alarm handler

Cyclic handlers are created by one of the following methods.

1) Creation by the system configuration file
   The static API “alarm_hand[]” described in the system configuration file creates a cyclic handler.
   Refer to “20.20 Alarm Handler Information (alarm_hand[])” for the details of “alarm_hand[]”.

2) Creation by cre_alm or acre_alm
   The cre_alm creates a cyclic handler with alarm handler ID indicated by parameter almid according to the content
   of parameter pk_calm.
   The acre_cyc creates a alarm handler according to the content of parameter pk_calm, and returns the created
   alarm handler ID.
   The information specified is shown below.
   - Alarm handler attribute (almatr)
     Only TA_HLNG can be specified for almatr.
   - Extended information (exinf)
   - Alarm handler start address (almhdr)

These service calls can be called from tasks that belong to Trusted Domain.
The following describes an example for coding acre_alm as a representative.

```c
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h"     /*Header file generated by cfg600px*/
extern void Almhdr1(VP_INT exinf);

#pragma task Task1  /*Refer to note*/
void Task1 (VP_INT exinf);  /*Refer to note*/
void Task1 (VP_INT exinf) {  
    ER almid;  /*Declares variable*/
    T_CALM pk_calm = {  /*Declares and initializes variable*/
        TA_HLNG,  /*Alarm handler attribute (cycatr)*/
        0,  /*Extended information (exinf)*/
        (FP)Almhdr1,  /*Start address (almhdr)*/
    };
    /* ......... */
    almid = acre_alm ( &pk_calm );  /*Create alarm handler*/
    /* ......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the
cfg600px generates these statement into the “kernel_id.h”.

RI600PX  CHAPTER 10  TIME MANAGEMENT FUNCTIONS
10.7.4 Delete alarm handler

- del alm

This service call deletes the alarm handler specified by parameter alm id.
This service call can be called from tasks that belong to Trusted Domain.

The following describes an example for coding this service call.

```c
#include "kernel.h"  /*Standard header file definition*/
#include "kernel_id.h"  /*Header file generated by cfg600px*/
#pragma task Task1  /*Refer to note*/
void Task1 (VP_INT exinf);  /*Refer to note*/

void Task1 (VP_INT exinf)
{
    ID almid = 8;  /*Declares and initializes variable*/

    /* .......... */

    ercd = del alm ( almid );  /*Delete alarm handler*/

    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.7.5 Start alarm handler operation

Moving to the operational state (STA state) is implemented by issuing the following service call from the processing program.

- `sta_alm, ista_alm`

  This service call sets the activation time of the alarm handler specified by `almid` in `almtim` (msec), and moves the alarm handler from the non-operational state (STP state) to operational state (STA state). As a result, the target alarm handler is handled as an activation target of the RI600PX. The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                   /*Refer to note 3*/
void Task1 (VP_INT exinf);          /*Refer to note 3*/

void Task1 (VP_INT exinf)
{
    ID almid = 1;              /*Declares and initializes variable*/
    /* .......... */
    sta_alm (almid);                /*Start alarm handler operation*/
    /* .......... */
}
```

**Note 1** When 0 is specified for `almtim`, the alarm handler will start at the next base clock interruption.

**Note 2** When the target alarm handler has already started (STA state), this service call sets the activation time of the target alarm handler in `almtim` (msec) after canceling the activation time.

**Note 3** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.7.6 Stop alarm handler operation

Moving to the non-operational state (STP state) is implemented by issuing the following service call from the processing program.

- **stp_alm, istp_alm**
  This service call moves the alarm handler specified by parameter cycid from the operational state (STA state) to non-operational state (STP state).
  As a result, the target alarm handler is excluded from activation targets of the RI600PX until issuance of **sta_alm** or **ista_alm**.
  The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma task Task1 /*Refer to note 2*/
void Task1 (VP_INT exinf); /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
  ID almid = 1; /*Declares and initializes variable*/
  /* ........... */
  stp_alm (almid); /*Stop alarm handler operation*/
  /* ........... */
}
```

**Note 1** This service call does not perform queuing of stop requests. If the target alarm handler has been moved to the non-operational state (STP state), therefore, no processing is performed but it is not handled as an error.

**Note 2** These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
10.7.7 Reference Alarm Handler State

A alarm handler status by issuing the following service call from the processing program.

- `ref_alm`, `iref_alm`
  Stores alarm handler state packet (current state, time until the next activation, etc.) of the alarm handler specified by parameter `cycid` in the area specified by parameter `pk_rcyc`.
  The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#define task(Task1)                  /*Refer to note 2*/

void Task1 (VP_INT exinf)          /*Refer to note 2*/
{
    ID almid = 1;              /*Declares and initializes variable*/
    T_RALM pk_ralm;                /*Declares data structure*/
    STAT almstat;                /*Declares variable*/
    RELTIM lefttim;                /*Declares variable*/

    /* .......... */
    ref_alm (almid, &pk_ralm);      /*Reference alarm handler state*/

    almstat = pk_ralm almstat;      /*Reference current state*/
    lefttim = pk_ralm lefttim;      /*Reference time left */

    /* .......... */
}
```

Note 1 For details about the alarm handler state packet, refer to "[Alarm handler state packet: T_RALM]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.8 System Time

10.8.1 Set system time

The system time can be set by issuing the following service call from the processing program. Note that even if the system time is changed, the actual time at which the time management requests made before that (e.g., task time-outs, task delay by dly_tsk, cyclic handlers, and alarm handlers) are generated will not change.

- set_tim, iset_tim

These service calls change the system time (unit: msec) to the time specified by parameter p_systim.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    SYSTIM p_systim;               /*Declares data structure*/
    p_systim.ltime = 3600;          /*Initializes data structure*/
    p_systim.utime = 0;             /*Initializes data structure*/
    /* .......... */
    set_tim (&p_systim);            /*Set system time*/
    /* .......... */
}
```

Note 1 For details about the system time packet SYSTIM, refer to "[System time packet: SYSTIM]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
10.8.2 Reference system time

The system time can be referenced by issuing the following service call from the processing program.

- `get_tim`, `iget_tim`

These service calls store the system time (unit: msec) into the area specified by parameter `p_systim`. The following describes an example for coding these service calls.

```c
#include "kernel.h"              /*Standard header file definition*/
#include "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);    /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    SYSTIM p_systim; /*Declares data structure*/
    UW ltime;       /*Declares variable*/
    UH utime;       /*Declares variable*/

    /* .......... */

    get_tim (&p_systim);    /*Reference System Time*/

    ltime = p_systim.ltime; /*Acquirer system time (lower 32 bits)*/
    utime = p_systim.utime; /*Acquirer system time (higher 16 bits)*/

    /* .......... */
}
```

Note 1 For details about the system time packet SYSTIM, refer to "[System time packet: SYSTIM]".

Note 2 These statements are unnecessary for the task which is created by the system configuration file because the `cfg600px` generates these statement into the "kernel_id.h".
10.9 Base Clock Timer Initialization Routine (_RI_init_cmt_knl( ))

10.9.1 User-own cording module

The base clock timer initialization routine must be implemented as user-own coding module.

Note: The source file for the base clock timer initialization routine provided by the RI600PX as a sample file is "init_cmt.c".

- Basic form of base clock timer initialization routine

The following shows the basic form of base clock timer initialization routine.

```c
#include    "kernel.h"    // Provided by RI600PX
#include    "kernel_id.h"  // Generated by cfg600px

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
#include    "ri_cmt.h"
#endif

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
#include    "kernel.h"
#endif

void _RI_init_cmt(void);

void _RI_init_cmt_knl(void)
{
#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
    _RI_init_cmt();
#endif
}
```

Note: The function name of the base clock timer initialization routine is "_RI_init_cmt_knl()".

- Description

The base clock timer initialization routine is called by vsta_knl and ivsta_knl. The "_RI_CLOCK_TIMER" is macro generated in "kernel_id.h" by the cfg600px. The definition value for "_RI_CLOCK_TIMER" is depend on Selection of timer channel for base clock (timer) in the system configuration file. Details are shown below.

<table>
<thead>
<tr>
<th>clock.timer</th>
<th>&quot;_RI_CLOCK_TIMER&quot; definition value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CMT0&quot;</td>
<td>0</td>
</tr>
<tr>
<td>&quot;CMT1&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;CMT2&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;CMT3&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;OTHER&quot;</td>
<td>0x7FFFFFFF</td>
</tr>
<tr>
<td>&quot;NOTIMER&quot;</td>
<td>-1</td>
</tr>
</tbody>
</table>

When "CMT0", "CMT1", "CMT2" or "CMT3" is specified for "clock.timer", the cfg600px generates the inline-function "void _RI_init_cmt(void)" for initializing the base clock timer. The _RI_init_cmt_knl() should be implemented only to call _RI_init_cmt().

When "OTHER" is specified for "clock.timer", the application must implement _RI_init_cmt_knl().
When "NOTIMER" is specified for "clock.timer", the _RI_init_cmt_knl() should be implemented so that nothing may be processed.

- Stack
  The base clock timer initialization routine uses the system stack.

- Service call
  The base clock timer initialization routine can issue service calls whose "Useful range" is "Non-task".

- PSW register when processing is started

  Table 10-3  PSW Register When Base Clock Timer Initialization Routine is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>IPL</td>
<td>Kernel interrupt mask level (system_IPL)</td>
<td>Do not lower IPL more than the start of processing.</td>
</tr>
<tr>
<td>PM</td>
<td>0</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>System stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 11 SYSTEM STATE MANAGEMENT FUNCTIONS

This chapter describes the system management functions performed by the RI600PX.

11.1 Outline

The RI600PX’s system status management function provides functions for referencing the system status such as the context type and CPU lock status, as well as functions for manipulating the system status such as ready queue rotation, scheduler activation, or the like.

Note, refer to “CHAPTER 15 SYSTEM DOWN” for system down (vsys_dwn, ivsys_dwn) and refer to “CHAPTER 17 SYSTEM INITIALIZATION” for starting of the RI600PX (vsta_knl, ivsta_knl).

11.2 Rotate Task Precedence

Task precedence is rotated by issuing the following service call from the processing program.

- rot_rdq, irot_rdq

This service call re-queues the first task of the ready queue corresponding to the priority specified by parameter tskpri to the end of the queue to change the task execution order explicitly.

The following shows the status transition when this service call is used.

Figure 11-1 Rotate Task Precedence
The following describes an example for coding these service calls.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/
#pragma cychandler Cychdr1 /*refer to note 5*/
void Cychdr1 (VP_INT exinf); /*refer to note 5*/
void Cychdr1 (VP_INT exinf) /*Cyclic handler*/
{
    PRI tskpri = 8; /*Declares and initializes variable*/
    /* .......... */
    irot_rdq (tskpri); /*Rotate task precedence*/
    /* .......... */
    return; /*Terminate cyclic handler*/
}
```

Note 1  This service call does not perform queuing of rotation requests. If no task is queued to the ready queue corresponding to the relevant priority, therefore, no processing is performed but it is not handled as an error.

Note 2  Round-robin scheduling can be implemented by issuing this service call via a cyclic handler in a constant cycle.

Note 3  The ready queue is a hash table that uses priority as the key, and tasks that have entered an executable state (READY state or RUNNING state) are queued in FIFO order. Therefore, the scheduler realizes the RI600PX's scheduling system by executing task detection processing from the highest priority level of the ready queue upon activation, and upon detection of queued tasks, giving the CPU use right to the first task of the proper priority level.

Note 4  When TPRI_SELF is specified as tskpri, the base priority of the invoking task is applied as the target priority of this service call. As for a task which has locked mutexes, the current priority might be different from the base priority. In this case, even if the task issues this service call specifying TPRI_SELF as parameter tskpri, the ready queue of the current priority that the invoking task belongs cannot be changed.

Note 5  These statements are unnecessary for the cyclic handler which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.
11.3 Reference Task ID in the RUNNING State

A RUNNING-state task is referenced by issuing the following service call from the processing program.

- `get_tid`, `iget_tid`

  These service calls store the ID of a task in the RUNNING state in the area specified by parameter `p_tskid`. The following describes an example for coding these service calls.

  ```c
  #include "kernel.h"              /* Standard header file definition */
  #include "kernel_id.h"           /* Header file generated by cfg600px */
  
  void Inthdr (void) /* Interrupt handler */
  {
      ID p_tskid; /* Declares variable */
      /* .......... */
      iget_tid (&p_tskid); /* Reference task ID in the RUNNING state */
      /* .......... */
      return; /* Terminate interrupt handler */
  }
  
  Note This service call stores TSK_NONE in the area specified by parameter `p_tskid` if no tasks that have entered the RUNNING state exist.
11.4 Lock and Unlock the CPU

In the CPU locked state, the task scheduling is prohibited, and kernel interrupts are masked. Therefore, exclusive processing can be achieved for all processing programs except non-kernel interrupt handlers.

The following service calls moves to the CPU locked state.

- loc_cpu, iloc_cpu
  These service calls transit the system to the CPU locked state.

The service calls that can be issued in the CPU locked state are limited to the one listed below.

<table>
<thead>
<tr>
<th>Service Call that can be issued</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_tsk</td>
<td>Terminate invoking task. (This service call transit the system to the CPU unlocked state.)</td>
</tr>
<tr>
<td>exd_tsk</td>
<td>Terminate and delete invoking task. (This service call transit the system to the CPU unlocked state.)</td>
</tr>
<tr>
<td>sns_tes</td>
<td>Reference task exception disabled state</td>
</tr>
<tr>
<td>loc_cpu, iloc_cpu</td>
<td>Lock the CPU.</td>
</tr>
<tr>
<td>unl_cpu, iunl_cpu</td>
<td>Unlock the CPU.</td>
</tr>
<tr>
<td>sns_loc</td>
<td>Reference CPU state.</td>
</tr>
<tr>
<td>sns_dsp</td>
<td>Reference dispatching state.</td>
</tr>
<tr>
<td>sns_ctx</td>
<td>Reference contexts.</td>
</tr>
<tr>
<td>sns_dpn</td>
<td>Reference dispatch pending state.</td>
</tr>
<tr>
<td>vsys_dwn, ivsys_dwn</td>
<td>System down</td>
</tr>
</tbody>
</table>

The following service calls and ext_tsk and exd_tsk release from the CPU locked state.

- unl_cpu, iunl_cpu
  These service calls transit the system to the CPU unlocked state.

The following shows a processing flow when using the CPU locked state.

Figure 11-2 Lock the CPU
The following describes an example for coding "lock the CPU" and "unlock the CPU".

```c
#include "kernel.h"  /* Standard header file definition*/
#include "kernel_id.h" /* Header file generated by cfg600px*/
#pragma task Task1 /* Refer to note 7*/
void Task1 (VP_INT exinf);  /* Refer to note 7*/
void Task1 (VP_INT exinf) 
{
    /* ......... */
    loc_cpu ();  /* Lock the CPU*/
    /* ......... */  /* CPU locked state*/
    unl_cpu ();  /* Unlock the CPU*/
    /* ......... */
}
```

**Note 1** The CPU locked state changed by issuing `loc_cpu` or `iloc_cpu` must be cancelled before the processing program that issued this service call ends.

**Note 2** The `loc_cpu` and `iloc_cpu` do not perform queuing of lock requests. If the system is in the CPU locked state, therefore, no processing is performed but it is not handled as an error.

**Note 3** The `unl_cpu` and `iunl_cpu` do not perform queuing of unlock requests. If the system is in the CPU unlocked state, therefore, no processing is performed but it is not handled as an error.

**Note 4** The `unl_cpu` and `iunl_cpu` do not cancel the dispatching disabled state that was set by issuing `dis_dsp`.

**Note 5** The base clock interrupt is masked during the CPU locked state. Therefore, time handled by the TIME MANAGEMENT FUNCTIONS may be delayed if the period of the CPU locked state becomes long.

**Note 6** For kernel interrupts, refer to “12.1 Interrupt Type”.

**Note 7** These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
11.5 Reference CPU Locked State

It may be necessary to refer to current CPU locked state in functions that are called from two or more tasks and handlers. In this case, `sns_loc` is useful.

- `sns_loc`
  This service call examines whether the system is in the CPU locked state or not. This service call returns TRUE when the system is in the CPU locked state, and return FALSE when the system is in the CPU unlocked state.

The following describes an example for coding this service call.

```c
#include "kernel.h" /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd; /*Declares variable*/
    /* .......... */
    ercd = sns_loc (); /*Reference CPU state*/

    if (ercd == TRUE) {
        /* .......... */ /*CPU locked state*/
    } else if (ercd == FALSE) {
        /* .......... */ /*CPU unlocked state*/
    }

    /* .......... */
}
```
11.6 Disable and Enable Dispatching

In the dispatching disabled state, the task scheduling is prohibited. Therefore, exclusive processing can be achieved for all tasks. The following service call moves to the dispatching disabled state. And also when PSW.IPL is changed to other than 0 by using \texttt{chg\_ims}, the system shifts to the dispatching disabled state.

- \texttt{dis\_dsp}
  
  This service call transits the system to the dispatching disabled state.

The dispatching disabled state is cancelled by the following service call, \texttt{ext\_tsk}, \texttt{exd\_tsk} and \texttt{chg\_ims} that changes PSW.IPL to 0.

- \texttt{ena\_dsp}
  
  This service call transits the system to the dispatching enabled state.

The following shows a processing flow when using the dispatching disabled state.

![Figure 11-3 Disable Dispatching](image)

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
void Task1 (VP_INT exinf);          /*Refer to note 5*/

void Task1 (VP_INT exinf)
{
    /* ......... */
    dis_dsp ();                     /*Disable dispatching*/
    /* ......... */                 /*Dispatching disabled state*/
    ena_dsp ();                     /*Enable dispatching*/
    /* ......... */
}
```

Note 1 The dispatching disabled state must be cancelled before the task that issued \texttt{dis\_dsp} moves to the DORMANT state.
Note 2 The \texttt{dis\_dsp} does not perform queuing of lock requests. If the system is in the dispatching disabled state, therefore, no processing is performed but it is not handled as an error.

Note 3 The \texttt{ena\_dsp} does not perform queuing of unlock requests. If the system is in the dispatching enabled state, therefore, no processing is performed but it is not handled as an error.

Note 4 If a service call (such as \texttt{wai\_sem, wai\_flg}) that may move the status of the invoking task is issued while the dispatching disabled state, that service call returns \texttt{E\_CTX} regardless of whether the required condition is immediately satisfied.

Note 5 These statements are unnecessary for the task which is created by the system configuration file because the \texttt{cfg600px} generates these statement into the “kernel\_id.h”.

### 11.7 Reference Dispatching Disabled State

It may be necessary to refer to current dispatching disabled state in functions that are called from two or more tasks. In this case, \texttt{sns\_dsp} is useful.

- **sns\_dsp**

  This service call examines whether the system is in the dispatching disabled state or not. This service call returns TRUE when the system is in the dispatching disabled state, and return FALSE when the system is in the dispatching enabled state.

  The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;                      /*Declares variable*/
    /* .......... */
    ercd = sns_dsp ();              /*Reference dispatching state*/
    if ( ercd == TRUE ) {
        /* .......... */
    } else if ( ercd == FALSE ) {
        /* .......... */
    }
    /* .......... */
}
```
11.8 Reference Context Type

It may be necessary to refer to current context type in functions that are called from two or more tasks and handlers. In this case, sns_ctx is useful.

- sns_ctx

This service call examines the context type of the processing program that issues this service call. This service call returns TRUE when the processing program is non-task context, and return FALSE when the processing program is task context.

The following describes an example for coding this service call.

```c
#include "kernel.h"        /*Standard header file definition*/
#include "kernel_id.h"      /*Header file generated by cfg600px*/

void CommonFunc ( void );
void CommonFunc ( void )
{
  BOOL ercd;                  /*Declares variable*/

  /* ............ */

  ercd = sns_ctx ( );        /*Reference context type*/

  if ( ercd == TRUE ) {
    /* ............ */
  } else if ( ercd == FALSE ) {
    /* ............ */
  }

  /* ............ */
}
```
11.9 Reference Dispatch Pending State

The state to fill either the following is called dispatch pending state.

- Dispatching disabled state
- CPU locked state
- PSW.IPL > 0, such as handlers

It may be necessary to refer to current dispatch pending state in functions that are called from two or more tasks and handlers. In this case, `sns_dpn` is useful.

- `sns_dpn`

  This service call examines whether the system is in the dispatch pending state or not. This service call returns TRUE when the system is in the dispatch pending state, and return FALSE when the system is not in the dispatch pending state.

  The following describes an example for coding this service call.

```c
#include    "kernel.h"    /*Standard header file definition*/
#include    "kernel_id.h"   /*Header file generated by cfg600px*/

void CommonFunc ( void );
void CommonFunc ( void )
{
    BOOL ercd;            /*Declares variable*/
    
    ercd = sns_dpn ();              /*Reference dispatch pending state*/

    if ( ercd == TRUE ) {
        /* ............ */          /*Dispatch pending state*/
    } else if ( ercd == FALSE ) {
        /* ............ */          /*Other state*/
    }

    /* ............ */
}
```
CHAPTER 12 INTERRUPT MANAGEMENT FUNCTIONS

This chapter describes the interrupt management functions performed by the RI600PX.

12.1 Interrupt Type

Interrupts are classified into kernel interrupt and non-kernel interrupt.

- Kernel interrupt
  An interrupt whose interrupt priority level is lower than or equal to the kernel interrupt mask level is called the kernel interrupt.
  A kernel interrupt handler can issue service calls.
  Note, however, that handling of kernel interrupts generated during kernel processing may be delayed until the interrupts become acceptable.

- Non-kernel interrupt
  An interrupt whose interrupt priority level is higher than the kernel interrupt mask level is called the non-kernel interrupt. The non-maskable interrupt is classified into non-kernel interrupt.
  A non-kernel interrupt handler must not issue service calls.
  Non-kernel interrupts generated during service-call processing are immediately accepted whether or not kernel processing is in progress.

  Note  The kernel interrupt mask level is defined by Kernel interrupt mask level (system_IPL) in System Information (system).

12.2 Fast Interrupt of the RX-MCU

The RX-MCU supports the “fast interrupt” function. Only one interrupt source can be made the fast interrupt. The fast interrupt is handled as the one that has interrupt priority level 15. To use the fast interrupt function, make sure there is only one interrupt source that is assigned interrupt priority level 15.
For the fast interrupt function to be used in the RI600PX, it is necessary that the interrupt concerned be handled as an non-kernel interrupt. In other words, the kernel interrupt mask level must be set to 14 or below.
And “os_int = NO;” and “pragma_switch = F;” are required for interrupt_vector[] definition.
And the FINTV register of the RX-MCU must be initialized to the start address of the handler in the Boot processing function (PowerON_Reset_PC( )).

12.3 CPU Exception

The following CPU exceptions are handled as non-kernel interrupt.

- Unconditional trap (INT, BRK instruction)
  Note, INT #1 to #8 are reserved by the RI600PX.

- Undefined instruction exception

- Privileged instruction exception

- Floating-point exception

On the other hand, the access exception handler is handled as kernel interrupt.
12.4 Base Clock Timer Interrupt

The TIME MANAGEMENT FUNCTIONS is realized by using base clock timer interrupts that occur at constant intervals. When the base clock timer interrupt occurs, the RI600PX's time management interrupt handler is activated and executes time-related processing (system time update, delayed wake-up/time-out of task, cyclic handler activation, etc.).

12.5 Multiple Interrupts

In the RI600PX, occurrence of an interrupt in an interrupt handler is called “multiple interrupts”. It can be set whether each interrupt handler for relocatable vector permits multiple interrupts. For details, refer to "20.21 Relocatable Vector Information (interrupt_vector[])".
12.6  Interrupt Handlers

The interrupt handler is a routine dedicated to interrupt servicing that is activated when an interrupt occurs. The RI600PX handles the interrupt handler as a non-task (module independent from tasks). Therefore, even if a task with the highest priority in the system is being executed, the processing is suspended when an interrupt occurs, and the control is passed to the interrupt handler.

12.6.1  Basic form of interrupt handlers

The following shows the basic form of interrupt handlers.

```
#include   "kernel.h"          /*Standard header file definition*/
#include   "kernel_id.h"         /*Header file generated by cfg600px*/

void Inthdr1 (void)
{
    /* ........ * /
    return;           /*Terminate interrupt handler*/
}
```

Note The cfg600px outputs the prototype declaration and #pragma interrupt directive for the handler function to kernel_id.h.

- Stack
   A interrupt handler uses the system stack.

- Service call
   The RI600PX handles the interrupt handler as a "non-task".
   The kernel interrupt handler can issue service calls whose "Useful range" is "Non-task".
   No service call can be issued in non-kernel interrupt handler.

Note If a service call (isig_sem, iset_flg, etc.) which causes dispatch processing (task scheduling processing) is issued in order to quickly complete the processing in the interrupt handler during the interval until the processing in the interrupt handler ends, the RI600PX executes only processing such as queue manipulation, counter manipulation, etc., and the actual dispatch processing is delayed until a return instruction is issued by the interrupt handler, upon which the actual dispatch processing is performed in batch.

- PSW register when processing is started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>- &quot;pragma_switch = E&quot;: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Other cases: 0</td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td>- Interrupt: Interrupt priority level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- CPU exception: Same before exception</td>
<td>Do not lower IPL more than the start of processing.</td>
</tr>
<tr>
<td>PM</td>
<td>0</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>System stack</td>
</tr>
<tr>
<td>C, Z, S, O</td>
<td>Undefined</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
12.6.2 Register interrupt handler

The RI600PX supports the static registration of interrupt handlers only. They cannot be registered dynamically by issuing a service call from the processing program.

Static interrupt handler registration means defining of interrupt handlers using static API "interrupt_vector[]" (relocatable vector) and "interrupt_fvector[]" (fixed vector/exception vector) in the system configuration file.

For details about the static API "interrupt_vector[]", refer to "20.21 Relocatable Vector Information (interrupt_vector[])", and for details about the static API "interrupt_fvector[]", refer to "20.22 Fixed Vector/Exception Vector Information (interrupt_fvector[])".

12.7 Maskable Interrupt Acknowledgement Status in Processing Programs

The maskable interrupt acknowledgement status of RX-M CU depends on the values of PSW.I and PSW.IPL. See the hardware manual for details.

The initial status is determined separately for each processing program. See Table 12-2 for details.

<table>
<thead>
<tr>
<th>Processing Program</th>
<th>PSW.I</th>
<th>PSW.IPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Task exception handling routine</td>
<td>1</td>
<td>Same as IPL in the task just before the task exception handling routine starts.</td>
</tr>
<tr>
<td>Cyclic handler, Alarm handler</td>
<td>1</td>
<td>Base clock interrupt priority level (IPL)</td>
</tr>
<tr>
<td>Interrupt Handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- &quot;pragma_switch = E&quot;: 1</td>
<td>1</td>
<td>- Interrupt: Interrupt priority level</td>
</tr>
<tr>
<td>- Other cases: 0</td>
<td>0</td>
<td>- CPU exception: Same before exception</td>
</tr>
</tbody>
</table>
12.8 Prohibit Maskable Interrupts

There is the following as a method of prohibiting maskable interrupts.

- Move to the CPU locked state by using loc_cpu, iloc_cpu
- Change PSW.IPL by using chg_ims, ichg_ims
- Change PSW.I and PSW.IPL directly (only for handlers)

12.8.1 Move to the CPU locked state by using loc_cpu, iloc_cpu

In the CPU locked state, PSW.IPL is changed to the Kernel interrupt mask level (system_IPL). Therefore, only kernel interrupts are prohibited in the CPU locked state.
Note, in the CPU locked state, service call issuance is restricted. For details, refer to "11.4 Lock and Unlock the CPU".

12.8.2 Change PSW.IPL by using chg_ims, ichg_ims

The PSW.IPL can be changed to arbitrary value by using chg_ims, ichg_ims.
When a task changes PSW.IPL to other than 0 by using chg_ims, the system is moved to the dispatching disabled state.
When a task returns PSW.IPL to 0, the system returns to the dispatching enabled state.
Do not issue ena_dsp while a task changes PSW.IPL to other than 0 by using chg_ims. If issuing ena_dsp, the system moves to the dispatching enabled state. If task dispatching occurs, PSW is changed for the dispatched task. Therefore PSW.IPL may be lowered without intending it.
The handlers must not lower PSW.IPL more than it starts.

12.8.3 Change PSW.I and PSW.IPL directly (only for handlers)

The handlers can change PSW.I and PSW.IPL directly. This method is faster than ichg_ims.
The handlers must not lower PSW.IPL more than it starts.
Note, the compiler provides following intrinsic functions for operating PSW. See CubeSuite+ RX Build User’s Manual for details about intrinsic functions.

- set_ipl(): Change PSW.IPL
- get_ipl(): Refer to PSW.IPL
- set_psw(): Change PSW
- get_psw(): Refer to PSW
CHAPTER 13  SYSTEM CONFIGURATION MANAGEMENT FUNCTIONS

This chapter describes the system configuration management functions performed by the RI600PX.

13.1 Outline

The RI600PX's system configuration management function provides the function to reference the version information.

13.2 Reference Version Information

The version information can be referenced by issuing the following service call from the processing program.

- ref_ver, iref_ver

These service calls store the version information into the area specified by parameter pk_rver.

The following describes an example for coding these service calls.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note 2*/
void Task1 (VP_INT exinf)          /*Refer to note 2*/
{
    T_RVER pk_rver;                /*Declares data structure*/
    UH maker;                      /*Declares variable*/
    UH prid;                       /*Declares variable*/

    /* ............... */

    ref_ver (&pk_rver);            /*Reference version information/ */

    maker = pk_rver.maker;         /*Acquirer system time (lower 32 bits)*/
    prid = pk_rver.prid;           /*Acquirer system time (higher 16 bits)*/

    /* ............... */
}
```

Note 1  For details about the version information packet T_RVER, refer to "[Version information packet: T_RVER]".

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
This chapter describes the object reset functions performed by the RI600PX.

14.1 Outline

The object reset function returns Data Queues, Mailboxes, Message Buffers, Fixed-Sized Memory Pools and Variable-Sized Memory Pools to the initial state. The object reset function falls outside μITRON4.0 specification.

14.2 Reset Data Queue

A data queue is reset by issuing the following service call from the processing program.

- vrst_dtq

This service call reset the data queue specified by parameter dtqid.

The data having been accumulated by the data queue area are annulled. The tasks to wait to send data to the target data queue are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

The following describes an example for coding this service call.

```c
#include    "kernel.h"          /*Standard header file definition*/
#include    "kernel_id.h"       /*Header file generated by cfg600px*/
#pragma task Task1              /*Refer to note 2*/
void Task1 (VP_INT exinf);     /*Refer to note 2*/

void Task1 (VP_INT exinf)
{
    ER  ercd;                  /*Declares variable*/
    ID  dtqid = 1;            /*Declares and initializes variable*/

    /* ........... */

    ercd = vrst_dtq ( dtqid ); /*Reset data queue*/

    /* ........... */
}
```

Note 1  In this service call, the tasks to wait to receive data do not released from the WAITING state.

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”.

14.3 Reset Mailbox

A mailbox is reset by issuing the following service call from the processing program.

- `vrst_mbx`

  This service call resets the mailbox specified by parameter `mbxid`.

  The messages having been accumulated by the mailbox come off from the management of the RI600PX.

  The following describes an example for coding this service call.

```c
#include "kernel.h"       /*Standard header file definition*/
#include "kernel_id.h"     /*Header file generated by cfg600px*/
#pragma task Task1         /*Refer to note 2*/
void Task1 (VP_INT exinf); /*Refer to note 2*/
void Task1 (VP_INT exinf)  
{
    ER ercd;               /*Declares variable*/
    ID mbxid = 1;          /*Declares and initializes variable*/
    /* ........... */
    ercd = vrst_mbx (mbxid); /*Reset mailbox*/
    /* ........... */
}
```

Note 1   In this service call, the tasks to wait to receive message do not released from the WAITING state.

Note 2   These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
14.4 Reset Message Buffer

A message buffer is reset by issuing the following service call from the processing program.

- vrst_mbf

This service call reset the message buffer specified by parameter mbfid.

The messages having been accumulated by the message buffer area are annulled. The tasks to wait to send message to the target message buffer are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

The following describes an example for coding this service call.

```c
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task
        Task1
        /*Refer to note 2*/
void Task1 (VP_INT exinf);          /*Refer to note 2*/
void Task1 (VP_INT exinf)
{
    ER  ercd;               /*Declares variable*/
    ID  mbfid = 1;          /*Declares and initializes variable*/
    /* .......... */
    ercd = vrst_mbf (mbfid); /*Reset message buffer*/
    /* .......... */
}
```

Note 1  In this service call, the tasks to wait to receive message do not released from the WAITING state.

Note 2  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
14.5 Reset Fixed-sized Memory Pool

A fixed-sized memory pool is reset by issuing the following service call from the processing program.

- vrst_mpf

This service call reset the fixed-sized memory pool specified by parameter mpfid.

The tasks to wait to get memory block from the target fixed-sized memory pool are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

All fixed-sized memory blocks that had already been acquired are returned to the target fixed-sized memory pool.

Therefore, do not access those fixed-sized memory blocks after issuing this service call.

The following describes an example for coding this service call.

```
#include    "kernel.h"              /*Standard header file definition*/
#include    "kernel_id.h"           /*Header file generated by cfg600px*/
#pragma task Task1                  /*Refer to note*/
void Task1 (VP_INT exinf);          /*Refer to note*/
void Task1 (VP_INT exinf)
{
    ER ercd;               /*Declares variable*/
    ID mpfid = 1;          /*Declares and initializes variable*/

    /* .......... */
    ercd = vrst_mpf (mpfid); /*Reset fixed-sized memory pool*/
    /* .......... */
}
```

Note These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the "kernel_id.h".
14.6 Reset Variable-sized Memory Pool

A variable-sized memory pool is reset by issuing the following service call from the processing program.

- **vrst_mpl**

  This service call resets the variable-sized memory pool specified by parameter *mpfid*. The tasks waiting to get memory blocks from the target variable-sized memory pool are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

  All variable-sized memory blocks that had already been acquired are returned to the target variable-sized memory pool. Therefore, do not access those variable-sized memory blocks after issuing this service call.

  The following describes an example for coding this service call.

  ```
  #include    "kernel.h"              /*Standard header file definition*/
  #include    "kernel_id.h"           /*Header file generated by cfg600px*/
  #pragma task Task1                  /*Refer to note*/
  void Task1 (VP_INT exinf);          /*Refer to note*/
  
  void Task1 (VP_INT exinf)
  {
      ER ercd;               /*Declares variable*/
      ID mplid = 1;          /*Declares and initializes variable*/
      /* .......... */
      ercd = vrst_mpl (mplid);  /*Reset variable-sized memory pool*/
      if (ercd == E_OK) {
          /* .......... */
      }
  }
  ```

  **Note**  These statements are unnecessary for the task which is created by the system configuration file because the cfg600px generates these statement into the “kernel_id.h”. 
CHAPTER 15 SYSTEM DOWN

This chapter describes the system down functions performed by the RI600PX.

15.1 Outline

When the event that cannot be recovered while the RI600PX is operating occurs, the system down is caused and the system down routine is invoked.

15.2 User-Own Coding Module

The system down routine must be implemented as user-own coding module.

Note  The source file for the system down routine provided by the RI600PX as a sample file is “sysdwn.c”.

15.2.1 System down routine (_RI_sys_dwn__( ))

The following shows the basic form of the system down routine. The system down routine must not return.

```c
#include "kernel.h"   /*Standard header file definition*/
#include "kernel_id.h" /*Header file generated by cfg600px*/

void _RI_sys_dwn___ ( W type, VW inf1, VW inf2, VW inf3 ); /*Prototype declaration*/

void _RI_sys_dwn___ ( W type, VW inf1, VW inf2, VW inf3 )
{
   /* ............ */
   while(1);
}
```

Note  The function name of the system down routine is "_RI_sys_dwn___".

- Stack
  The system down routine uses the system stack.

- Service call
  The system down routine must not issue service calls.
15.2.2 Parameters of system down routine

- type == -1 (Error when a kernel interrupt handler ends)

Table 15-2 Parameters of System Down Routine (type == -1)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>undefined</td>
<td>PSW.PM == 1 (user mode) when a kernel interrupt handler ends.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>undefined</td>
<td>PSW.IPL &gt; kernel interrupt mask level when a kernel interrupt handler ends.</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>undefined</td>
<td>The system is in the CPU locked state when a kernel interrupt handler ends.</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>undefined</td>
<td>Stack pointer points for the interrupted task points out of user stack for the task.</td>
</tr>
</tbody>
</table>

- type == -2 (Error in ext_tsk)

Table 15-3 Parameters of System Down Routine (type == -2)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>undefined</td>
<td>The ext_tsk is called in the non-task context.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>undefined</td>
<td>The ext_tsk is called in the state &quot;PSW.IPL &gt; Kernel interrupt mask level&quot;.</td>
</tr>
</tbody>
</table>

Table 15-1 PSW Register When System Down Routine is Started

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
| IPL  | - type < 0 : Undefined  
- type >= 0 : Same before system down | Do not lower IPL more than the start of processing. |
| PM   | 0     | Supervisor mode                                                     |
| U    | 0     | System stack                                                        |
| C, Z, S, O | undefined |                                                     |
| Others | 0     |                                                                     |
- **type == -3 (Unlinked service call issued)**

  **Table 15-4 Parameters of System Down Routine (type == -3)**

<table>
<thead>
<tr>
<th>inf1 inf2 inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOSPT (-9)</td>
<td>undefined undefined Unlinked service call is issued.</td>
</tr>
</tbody>
</table>

Note Refer to “2.6.1 Service call information files and "-ri600_preinit_mrc" compiler option”.

- **type == -4 (Error at returning from task exception handling routine)**

  **Table 15-5 Parameters of System Down Routine (type == -4)**

<table>
<thead>
<tr>
<th>inf1 inf2 inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX (-25) 7</td>
<td>undefined A task exception handling routine returns in the state “PSW.IPL &gt; Kernel interrupt mask level”.</td>
</tr>
<tr>
<td>8 undefined</td>
<td>A task exception handling routine returns in the CPU locked state.</td>
</tr>
<tr>
<td>9 undefined</td>
<td>A task exception handling routine returns in the non-task context.</td>
</tr>
</tbody>
</table>

- **type == -5 (Error in exd_tsk)**

  **Table 15-6 Parameters of System Down Routine (type == -5)**

<table>
<thead>
<tr>
<th>inf1 inf2 inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX (-25) 10</td>
<td>undefined The exd_tsk is called in the state “PSW.IPL &gt; Kernel interrupt mask level”.</td>
</tr>
<tr>
<td>11 undefined</td>
<td>The exd_tsk is called in the non-task context.</td>
</tr>
</tbody>
</table>
- **type** == -6 (Error in *vsta_knl* and *ivsta_knl*)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_PAR (-17)</td>
<td>15</td>
<td>undefined</td>
<td>Error regarding to registration of memory object (memory_object[])</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 ) Start address is not 16-bytes boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 ) Bit15 of either acptn1, acptn2 or acptn3 is 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 ) acptn1 == acptn2 == acptn3 == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 ) Bits corresponded to the domain ID that is larger than the maximum domain ID (VTMAX_DOMAIN) of either acptn1, acptn2 or acptn3 is 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 ) Start address &gt; termination address</td>
</tr>
<tr>
<td>E_OBJ (-41)</td>
<td>undefined</td>
<td></td>
<td>Multiple memory object with the same address are registered.</td>
</tr>
<tr>
<td>E_OACV (-27)</td>
<td>undefined</td>
<td></td>
<td>The number of memory objects from which the access is permitted to one domain exceeds 7.</td>
</tr>
<tr>
<td>E_PAR (-17)</td>
<td>16</td>
<td>undefined</td>
<td>Error regarding to task creation (task[])</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The “termination address of user stack + 1” is not 16-bytes boundary.</td>
</tr>
</tbody>
</table>

- **type** == -16 (Undefined relocatable vector interrupt)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PC, which is pushed to the stack by CPU’s interrupt operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PSW, which is pushed to the stack by CPU’s interrupt operation</td>
</tr>
</tbody>
</table>

- **type** == -17 (Undefined fixed vector/exception vector interrupt)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PC, which is pushed to the stack by CPU’s interrupt operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PSW, which is pushed to the stack by CPU’s interrupt operation</td>
</tr>
</tbody>
</table>
- type > 0 (Issuing vsys_dwn, ivsys_dwn from application))
  0 and a negative type value is reserved by the RI600PX. When calling vsys_dwn, ivsys_dwn from application, use positive type value.

Table 15-10  Parameters of System Down Routine (type > 0)

<table>
<thead>
<tr>
<th>inf1</th>
<th>inf2</th>
<th>inf3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value specified for vsys_dwn, ivsys_dwn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 16 SCHEDULING FUNCTION

This chapter describes the scheduler of the RI600PX.

16.1 Outline

The scheduling functions provided by the RI600PX consist of functions manage/decide the order in which tasks are executed by monitoring the transition states of dynamically changing tasks, so that the CPU use right is given to the optimum task.

16.2 Processing Unit and Precedence

An application program is executed in the following processing units.

- Task (including task exception handling routine)
- Interrupt handler
- Cyclic handler
- Alarm handler
- Access exception handler

The various processing units are processed in the following order of precedence.

1) Interrupt handlers, cyclic handlers, alarm handlers
2) Access exception handler
3) Scheduler
4) Tasks (including task exception handling routines)

The "scheduler" is the RI600PX’s processing that schedules running task and dispatches to the task. Since interrupt handler, cyclic handlers, alarm handlers and access exception handler have higher precedence than the scheduler, no tasks and no task exception handling routines are executed while these handlers are executing. (Refer to "16.7 Task Scheduling in Non-Tasks").

The precedence of an interrupt handler becomes higher when the interrupt level is higher.
The precedence of a cyclic handler and alarm handler is the same as the interrupt handler which interrupt level is same as the base clock timer interrupt.
The order of precedence for tasks depends on the current priority of the tasks.

16.3 Task Drive Method

The RI600PX employs the Event-driven system in which the scheduler is activated when an event (trigger) occurs.

- Event-driven system
  Under the event-driven system of the RI600PX, the scheduler is activated upon occurrence of the events listed below and dispatch processing (task scheduling processing) is executed.
    - Issuance of service call that may cause task state transition
    - Issuance of instruction for returning from non-task (cyclic handler, interrupt handler, etc.)
    - Occurrence of base clock interrupt used when achieving TIME MANAGEMENT FUNCTIONS
16.4 Task Scheduling Method

As task scheduling methods, the RI600PX employs the Priority level method, which uses the priority level defined for each task, and the FCFS method, which uses the time elapsed from the point when a task becomes target to RI600PX scheduling.

- Priority level method
  A task with the highest current priority is selected from among all the tasks that have entered an executable state (RUNNING state or READY state), and given the CPU use right.

- FCFS method
  When two or more “task with the highest priority level” exist, the scheduling target task can not be decided only by the Priority level method. In this case, the RI600PX decides the scheduling target task by first come first served (FCFS) method. Concretely, the task that enters to executable state (READY state) earliest among them, and given the CPU use right.

16.4.1 Ready queue

The RI600PX uses a “ready queue” to implement task scheduling. The ready queue is a hash table that uses priority as the key, and tasks that have entered an executable state (READY state or RUNNING state) are queued in FIFO order. Therefore, the scheduler realizes the RI600PX’s scheduling method (priority level or FCFS) by executing task detection processing from the highest priority level of the ready queue upon activation, and upon detection of queued tasks, giving the CPU use right to the first task of the proper priority level. The following shows the case where multiple tasks are queued to a ready queue.

Figure 16-1 Implementation of Scheduling Method (Priority Level Method or FCFS Method)

- Create ready queue
  In the RI600PX, the method of creating a ready queue is limited to “static creation”. Ready queues therefore cannot be created dynamically using a method such as issuing a service call from a processing program. Static ready queue creation means defining of Maximum task priority (priority) in System Information (system) in the system configuration file.
16.5 Task Scheduling Lock Function

The RI600PX provides the scheduling lock function for manipulating the scheduler status explicitly from the processing program and disabling/enabling dispatch processing. The following shows a processing flow when using the scheduling lock function.

![Figure 16-2 Scheduling Lock Function](image)

For details, refer to "11.4 Lock and Unlock the CPU" and "11.6 Disable and Enable Dispatching".
16.6 Idling

When there is no RUNNING or READY task, the RI600PX enters an endless loop and waits for interrupts.

16.7 Task Scheduling in Non-Tasks

If processing of non-tasks starts, any tasks will not be performed until non-task processing is completed, since the precedence of non-task (interrupt handler, cyclic handler, alarm handler and access exception handler) is higher than task as shown in “16.2 Processing Unit and Precedence”.

The following shows an example when a service call accompanying dispatch processing is issued in non-tasks. In this example, when the interrupt handler issues iwup_tsk, the Task A whose priority is higher than the task B is released from the WAITING state, but processing of the interrupt handler is continued at this time, without performing the task A yet. When processing of the interrupt handler is completed, the scheduler is started, and as a result, the task A is performed.

Figure 16-3 Scheduling in Non-Tasks
CHAPTER 17 SYSTEM INITIALIZATION

This chapter describes the system initialization routine performed by the RI600PX.

17.1 Outline

The following shows a processing flow from when a reset interrupt occurs until the control is passed to the task.

Figure 17-1 Processing Flow (System Initialization)
17.2 Boot Processing File (User-Own Coding Module)

The following should be described in the boot processing file.

1) Boot processing function (PowerON_Reset_PC( ))

2) Include kernel_ram.h and kernel_rom.h

Note The boot processing file which is provided by the RI600PX as a sample file is “resetprg.c”.

17.2.1 Boot processing function (PowerON_Reset_PC( ))

The boot processing function is the program registered in the reset vector, and is executed in supervisor mode. Generally, following processing are required in the boot processing function.

- Initialize the processor and hardwares
  
  If using Fast Interrupt of the RX-MCU, initialize the FINTV register to the start address of the fast interrupt handler.

- Initialize C/C++ runtime environment (Initialize sections, etc.)

- Start the RI600PX (call vsta_knl or ivsta_knl)

- Basic form of boot processing function

  The boot processing function should be implemented as “void PowerON_Reset_PC(void)”. When the name of the boot processing function is other, it is necessary to define the function name to “interrupt_fvector[31]” in the system configuration file.

Note For the details of the details of the static API “interrupt_fvector[]”, refer to “20.22 Fixed Vector/Exception Vector Information (interrupt_fvector[])”.

- The points of concern about the boot processing function

  - Stack
    
    Describe #pragma entry directive to be shown below. Thereby, the object code which sets the stack pointer (ISP) as the system stack at the head of the boot processing function is generated.

    ```
    #pragma entry PowerON_Reset_PC
    ```

  - PSW register
    
    Keep the status that all interrupts are prohibited and in the supervisor mode until calling the Kernel Initialization Module (vsta_knl, ivsta_knl). This status is satisfied just behind CPU reset (PSW.I=0, PSW.PM=0). Generally, the boot processing function should not change the PSW.

  - EXTB register (RXv2 architecture)
    
    Initialize EXTB register to the start address of FIX_INTERRUPT_VECTOR section if needed. Please refer to “FIX_INTERRUPT_VECTOR section” in section 2.6.4.

  - Service call
    
    Since the boot processing function is executed before executing of Kernel Initialization Module (vsta_knl, ivsta_knl), service calls except vsta_knl and ivsta_knl must not be called from the boot processing function.
17.2.2 Include kernel_ram.h and kernel_rom.h

The boot processing file must include "kernel_ram.h" and "kernel_rom.h", which are generated by the cfg600px, in this order.

17.2.3 Compiler option for boot processing file

The following compiler options are required for the boot processing file.

- "-lang=c" or "-lang=c99"
- "-nostuff"
- Suitable "-isa" or "-cpu"

Note Compiler option "-isa" is supported by the compiler CC-RX V2.01 or later.
17.2.4 Example of the boot processing file

```c
#include <machine.h>
#include <_h_c_lib.h>
#include "typedefine.h"  // Define Types
#include "kernel.h"      // Provided by RI600PX
#include "kernel_id.h"    // Generated by cfg600px

#ifdef __cplusplus
extern "C" {
#endif

void PowerON_Reset_PC(void);

#ifdef __cplusplus
}
#endif

#define FPSW_init 0x00000000  // FPSW bit base pattern

#ifdef __cplusplus
extern "C" {
#endif

void srand(_UINT);     // Remove the comment when you use rand()
extern _SBYTE *__slpstr; // Remove the comment when you use strtok()

#ifdef __cplusplus
extern void _INIT_IOLIB(void);
#endif

extern void _CLOSEALL(void);

#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C" {                  // Sections C$INIT and C$END will be generated
#endif

extern void _CALL_INIT(void);
extern void _CALL_END(void);

#ifdef __cplusplus
}
#endif
```
// Section definition
#pragma section P PS
#pragma section B BS
#pragma section C CS
#pragma section D DS
#pragma entry PowerON_Reset_PC

// Boot processing
void PowerON_Reset_PC(void)
{
    #ifdef __ROZ                        // Initialize FPSW
        #define _ROUND 0x00000001           // Let FPSW RMbits=01 (round to zero)
    #else
        #define _ROUND 0x00000000           // Let FPSW RMbits=00 (round to nearest)
    #endif
    #ifdef __DOFF
        #define _DENOM 0x00000100           // Let FPSW DNbit=1 (denormal as zero)
    #else
        #define _DENOM 0x00000000           // Let FPSW DNbit=0 (denormal as is)
    #endif

    //    set_extb(__sectop("FIX_INTERRUPT_VECTOR")); // Initialize EXTB register
    //  (only for RXv2 arch.)
    set_fpsw(FPSW_init | _ROUND | _DENOM);
    _INITSCT();                     // Initialize Sections
    //  _INIT_IOLIB();                  // Use SIM I/O
    //  errno=0;                        // Remove the comment when you use errno
    //  srand((_UINT)1);                // Remove the comment when you use rand()
    //  _slptr=NULL;                    // Remove the comment when you use strtok()
    //  HardwareSetup();                // Use Hardware Setup
    nop();

    // set_fintv(<handler address>); // Initialize FINTV register

    // _CALL_INIT();                   // Remove the comment when you use global class object
    vsta_knl();                       // Start RI600PX
    //  Never return from vsta_knl

    // _CLOSEALL();                    // Use SIM I/O

    // _CALL_END();                    // Remove the comment when you use global class object
    brk();
}

// RI600PX system data
#include "kernel_ram.h"     // generated by cfg600px
#include "kernel_rom.h"     // generated by cfg600px
17.3 Kernel Initialization Module (vsta_knl, ivsta_knl)

The kernel initialization module is executed by calling vsta_knl, ivsta_knl. Generally, vsta_knl, ivsta_knl is called from the Boot processing function (PowerON_Reset_PC()). The following processing is executed in the kernel initialization module.

1) Initialize ISP register to the end address of SI section + 1
2) Initialize INTB register to the start address of the relocatable vector table (INTERRUPT_VECTOR section). The relocatable vector table is generated by the cfg600px.
3) Initialize the system time to 0.
4) Create various object which are defined in the system configuration file. If an error is detected in this process, the system will be down.
5) Initialize MPU (Memory Protection Unit).
6) Initialize base clock timer (call Base Clock Timer Initialization Routine (_RI_init_cmt_knl( )))
7) Pass control to scheduler
17.4 Section Initialization Function (_INITSCT( ))

The section initialization function "_INITSCT()" called from Boot processing function (PowerON_Reset_PC( )) is provided by the compiler. The _INITSCT() clears the uninitialized data section to 0 and initializes the initialized data section in order to the tables described in the Section information file (User-Own Coding Module).

The user needs to write the sections to be initialized to the tables for section initialization (DTBL and BTBL) in the section information file. The section address operator is used to set the start and end addresses of the sections used by the _INITSCT(). Section names in the section initialization tables are declared, using C$BSEC for uninitialized data areas, and C$DSEC for initialized data areas.

Initialized sections written in DTBL must be mapped from ROM to RAM by using "-rom" linker option. For details, refer to "2.6.5 Initialized data section".

### 17.4.1 Section information file (User-Own Coding Module)

The section information file should be implemented as user-owned coding module. The example of the section information file is shown below.

**Note**  The section information file which is provided by the RI600PX as a sample file is “dbsct.c”.

```c
#include "typedefine.h"

#pragma unpack

#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s; /* Start address of the initialized data section in ROM */
    _UBYTE *rom_e; /* End address of the initialized data section in ROM */
    _UBYTE *ram_s; /* Start address of the initialized data section in RAM */
}
_DTBL[] = {
    {_sectop("D"), _secend("D"), _sectop("R") },
    {_sectop("D_2"), _secend("D_2"), _sectop("R_2") },
    {_sectop("D_1"), _secend("D_1"), _sectop("R_1") },
    {_sectop("DS"), _secend("DS"), _sectop("RS") },
    {_sectop("DS_2"), _secend("DS_2"), _sectop("RS_2") },
    {_sectop("DS_1"), _secend("DS_1"), _sectop("RS_1") },
    {_sectop("DU_SH"), _secend("DU_SH"), _sectop("RU_SH") },
    {_sectop("DU_SH_2"), _secend("DU_SH_2"), _sectop("RU_SH_2") },
    {_sectop("DU_USB_1"), _secend("DU_USB_1"), _sectop("RU_USB_1") },
    {_sectop("DU_MASTERDOM"), _secend("DU_MASTERDOM"),
        _sectop("RU_MASTERDOM") },
    {_sectop("DU_MASTERDOM_2"), _secend("DU_MASTERDOM_2"),
        _sectop("RU_MASTERDOM_2") },
    {_sectop("DU_MASTERDOM_1"), _secend("DU_MASTERDOM_1"),
        _sectop("RU_MASTERDOM_1") },
    {_sectop("DU_DOM_A"), _secend("DU_DOM_A"), _sectop("RU_DOM_A") },
    {_sectop("DU_DOM_A_2"), _secend("DU_DOM_A_2"), _sectop("RU_DOM_A_2") },
    {_sectop("DU_DOM_A_1"), _secend("DU_DOM_A_1"), _sectop("RU_DOM_A_1") },
    {_sectop("DU_DOM_B"), _secend("DU_DOM_B"), _sectop("RU_DOM_B") },
    {_sectop("DU_DOM_B_2"), _secend("DU_DOM_B_2"), _sectop("RU_DOM_B_2") },
    {_sectop("DU_DOM_B_1"), _secend("DU_DOM_B_1"), _sectop("RU_DOM_B_1") }
};

#pragma section C C$BSEC
extern const struct {
    _UBYTE *b_s; /* Start address of non-initialized data section */
    _UBYTE *b_e; /* End address of non-initialized data section */
}
_BTBL[] = {
    {_sectop("B"), _secend("B") },
    {_sectop("B_2"), _secend("B_2") },
    {_sectop("B_1"), _secend("B_1") },
    {_sectop("BS"), _secend("BS") },
    {_sectop("BS_2"), _secend("BS_2") },
    {_sectop("BS_1"), _secend("BS_1") }
};
```
{ __sectop("BU_SH"), __secend("BU_SH") },
{ __sectop("BU_SH_2"), __secend("BU_SH_2") },
{ __sectop("BU_SH_1"), __secend("BU_SH_1") },
{ __sectop("DU_MASTERDOM"), __secend("DU_MASTERDOM")},
{ __sectop("DU_MASTERDOM_2"), __secend("DU_MASTERDOM_2")},
{ __sectop("DU_MASTERDOM_1"), __secend("DU_MASTERDOM_2") },
{ __sectop("DU_DOM_A"), __secend("DU_DOM_A") },
{ __sectop("DU_DOM_A_2"), __secend("DU_DOM_A_2") },
{ __sectop("DU_DOM_A_1"), __secend("DU_DOM_A_1") },
{ __sectop("DU_DOM_B"), __secend("DU_DOM_B") },
{ __sectop("DU_DOM_B_2"), __secend("DU_DOM_B_2") },
{ __sectop("DU_DOM_B_1"), __secend("DU_DOM_B_1") },
{ __sectop("DU_DOM_B_1"), __secend("DU_DOM_B_1") },
{ __sectop("DU_DOM_A"), __secend("DU_DOM_A") },
{ __sectop("DU_DOM_A_2"), __secend("DU_DOM_A_2") },
{ __sectop("DU_DOM_A_1"), __secend("DU_DOM_A_1") },
{ __sectop("DU_DOM_B"), __secend("DU_DOM_B") },
{ __sectop("DU_DOM_B_2"), __secend("DU_DOM_B_2") },
{ __sectop("DU_DOM_B_1"), __secend("DU_DOM_B_1") };

#pragma section C CSSBSEC
extern const struct {
    _UBYTE *b_s;         /* Start address of non-initialized data section */
    _UBYTE *b_e;         /* End address of non-initialized data section */
}   _BTBL[] = {
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") },
    { __sectop("BS"), __secend("BS") },
    { __sectop("BS_2"), __secend("BS_2") },
    { __sectop("BS_1"), __secend("BS_1") },
    { __sectop("BU_SH"), __secend("BU_SH") },
    { __sectop("BU_SH_2"), __secend("BU_SH_2") },
    { __sectop("BU_SH_1"), __secend("BU_SH_1") },
    { __sectop("BU_MASTERDOM"), __secend("BU_MASTERDOM") },
    { __sectop("BU_MASTERDOM_2"), __secend("BU_MASTERDOM_2") },
    { __sectop("BU_MASTERDOM_1"), __secend("BU_MASTERDOM_1") },
    { __sectop("BU_DOM_A"), __secend("BU_DOM_A") },
    { __sectop("BU_DOM_A_2"), __secend("BU_DOM_A_2") },
    { __sectop("BU_DOM_A_1"), __secend("BU_DOM_A_1") },
    { __sectop("BU_DOM_B"), __secend("BU_DOM_B") },
    { __sectop("BU_DOM_B_2"), __secend("BU_DOM_B_2") },
    { __sectop("BU_DOM_B_1"), __secend("BU_DOM_B_1") }
};

#pragma section

/*
** CTBL prevents excessive output of L1100 messages when linking.
** Even if CTBL is deleted, the operation of the program does not change.
*/
_UBYTE * const _CTBL[] = {
    { __sectop("C_1"), __sectop("C_2"), __sectop("C"),
    { __sectop("W_1"), __sectop("W_2"), __sectop("W") },
    { __sectop("CU_MASTERDOM_1"), __sectop("CU_MASTERDOM_2") },
    { __sectop("CU_DOM_A_1"), __sectop("CU_DOM_A_2") },
    { __sectop("CU_DOM_B_1"), __sectop("CU_DOM_B_2") },
    { __sectop("WU_SH_1") },
    { __sectop("LU_SH") },
    { __sectop("CU_SH_1") },
    { __sectop("CU_SH_2") },
    { __sectop("CS_1") },
    { __sectop("CS_2") },
    { __sectop("CS") },
    { __sectop("P") }
};
#pragma packoption
17.5 Registers in Fixed Vector Table / Exception Vector Table

For some MCUs, the endian select register, ID code protection on connection of the on-chip debugger, etc. are assigned in the address from 0xFFFFFFFF80 to 0xFFFFFFFFBF in fixed vector table (RXv1 architecture) / exception vector table (RXv2 architecture). To set up such registers, describe "interrupt_fvector[]" in the system configuration file. For details, refer to "20.22 Fixed Vector/Exception Vector Information (interrupt_fvector[])".
CHAPTER 18 DATA TYPES AND MACROS

This chapter describes the data types and macros, which are used when issuing service calls provided by the RI600PX.

Note  \(<ri\_root>\) indicates the installation folder of RI600PX. The default folder is “C:\Program Files\Renesas Electronics\CubeSuite+\RI600PX”.

18.1 Data Types

The following lists the data types of parameters specified when issuing a service call. Macro definition of the data type is performed by \(<ri\_root>\in600\kernel.h\), or \(<ri\_root>\inc600\itron.h\) that is included by \kernel.h\.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>signed char</td>
<td>Signed 8-bit integer</td>
</tr>
<tr>
<td>H</td>
<td>signed short</td>
<td>Signed 16-bit integer</td>
</tr>
<tr>
<td>W</td>
<td>signed long</td>
<td>Signed 32-bit integer</td>
</tr>
<tr>
<td>D</td>
<td>signed long long</td>
<td>Signed 64-bit integer</td>
</tr>
<tr>
<td>UB</td>
<td>unsigned char</td>
<td>Unsigned 8-bit integer</td>
</tr>
<tr>
<td>UH</td>
<td>unsigned short</td>
<td>Unsigned 16-bit integer</td>
</tr>
<tr>
<td>UW</td>
<td>unsigned long</td>
<td>Unsigned 32-bit integer</td>
</tr>
<tr>
<td>UD</td>
<td>unsigned long long</td>
<td>Unsigned 64-bit integer</td>
</tr>
<tr>
<td>VB</td>
<td>signed char</td>
<td>8-bit value with unknown data type</td>
</tr>
<tr>
<td>VH</td>
<td>signed short</td>
<td>16-bit value with unknown data type</td>
</tr>
<tr>
<td>VW</td>
<td>signed long</td>
<td>32-bit value with unknown data type</td>
</tr>
<tr>
<td>VD</td>
<td>signed long long</td>
<td>64-bit value with unknown data type</td>
</tr>
<tr>
<td>VP</td>
<td>void *</td>
<td>Pointer to unknown data type</td>
</tr>
<tr>
<td>FP</td>
<td>void ((*))</td>
<td>Processing unit start address (pointer to a function)</td>
</tr>
<tr>
<td>INT</td>
<td>signed long</td>
<td>Signed 32-bit integer</td>
</tr>
<tr>
<td>UINT</td>
<td>unsigned long</td>
<td>Unsigned 32-bit integer</td>
</tr>
<tr>
<td>BOOL</td>
<td>signed long</td>
<td>Boolean value (TRUE or FALSE)</td>
</tr>
<tr>
<td>ER</td>
<td>signed long</td>
<td>Error code</td>
</tr>
<tr>
<td>ID</td>
<td>signed short</td>
<td>Object ID</td>
</tr>
<tr>
<td>ATR</td>
<td>unsigned short</td>
<td>Object attribute</td>
</tr>
<tr>
<td>STAT</td>
<td>unsigned short</td>
<td>Object state</td>
</tr>
<tr>
<td>MODE</td>
<td>unsigned short</td>
<td>Service call operational mode</td>
</tr>
<tr>
<td>PRI</td>
<td>signed short</td>
<td>Priority for tasks or messages</td>
</tr>
<tr>
<td>SIZE</td>
<td>unsigned long</td>
<td>Memory area size (in bytes)</td>
</tr>
<tr>
<td>TMO</td>
<td>signed long</td>
<td>Time-out (in millisecond)</td>
</tr>
<tr>
<td>RELTIM</td>
<td>unsigned long</td>
<td>Relative time (in millisecond)</td>
</tr>
</tbody>
</table>
18.2 Constant macros

The following lists the constant macros.

The constant macros are defined by either of following header files.

- `<ri_root>/inc600/kernel.h`
- `<ri_root>/inc600/itron.h`, which is included by kernel.h
- System information header file `kernel_id.h`, which is generated by the `cfg600px`
  The contents of this file is changed according to the system configuration file.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Macro</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>NULL</td>
<td>0</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>TRUE</td>
<td>1</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>0</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_OK</td>
<td>0</td>
<td>itron.h</td>
</tr>
<tr>
<td>Classification</td>
<td>Macro</td>
<td>Definition</td>
<td>Where</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>Attribute</td>
<td>TA_NULL</td>
<td>0</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>TA_HLNG</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_TFIFO</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_TPRI</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_MFIFO</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_MPRI</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_ACT</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_WSGL</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_WMUL</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_CLR</td>
<td>0x0004</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_CEILING</td>
<td>0x0003</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_STA</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TA_PHS</td>
<td>0x0004</td>
<td>kernel.h</td>
</tr>
<tr>
<td>Time-out</td>
<td>TMO_POL</td>
<td>0</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>TMO_FEVR</td>
<td>-1</td>
<td>itron.h</td>
</tr>
<tr>
<td>Operation mode</td>
<td>TWF_ANDW</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TWF_ORW</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td>Task exception</td>
<td>TTEX_ENA</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTEX_DIS</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td>Classification</td>
<td>Macro</td>
<td>Definition</td>
<td>Where</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>Object state</strong></td>
<td>TTS_RUN</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTS_RDY</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTS_WAI</td>
<td>0x0004</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTS_SUS</td>
<td>0x0008</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTS_WAS</td>
<td>0x000C</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTS_DMT</td>
<td>0x0010</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_SLP</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_DLY</td>
<td>0x0002</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_SEM</td>
<td>0x0004</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_FLG</td>
<td>0x0008</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_SDTQ</td>
<td>0x0010</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_RDTQ</td>
<td>0x0020</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_MBX</td>
<td>0x0040</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_MTX</td>
<td>0x0080</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_SMBF</td>
<td>0x0100</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_RMBF</td>
<td>0x0200</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_MPF</td>
<td>0x2000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TTT_MPL</td>
<td>0x4000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TCYC_STP</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TCYC_STA</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TALM_STP</td>
<td>0x0000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TALM_STA</td>
<td>0x0001</td>
<td>kernel.h</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>TSK_SELF</td>
<td>0</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TSK_NONE</td>
<td>0</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TPRI_SELF</td>
<td>0</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TPRI_INI</td>
<td>0</td>
<td>kernel.h</td>
</tr>
<tr>
<td>Classification</td>
<td>Macro</td>
<td>Definition</td>
<td>Where</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Kernel</td>
<td>TMIN_TPRI</td>
<td>1</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_TPRI</td>
<td>system.priority</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>TMIN_MPRI</td>
<td>1</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_MPRI</td>
<td>system.message_pri</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>TKERNEL_MAKER</td>
<td>0x011B</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TKERNEL_PRID</td>
<td>0x0004</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TKERNEL_SPVER</td>
<td>0x5403</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TKERNEL_PRVER</td>
<td>0x0120</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_ACTCNT</td>
<td>255</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_WUPCNT</td>
<td>255</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_SUSCNT</td>
<td>1</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TBIT_FLGPTN</td>
<td>32</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TBIT_TEXPTN</td>
<td>32</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TIC_NUME</td>
<td>system.tic_nume</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>TIC_DENO</td>
<td>system.tic_deno</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>TMAX_MAXSEM</td>
<td>65535</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_DOMAIN</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_TSK</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_SEM</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_FLG</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_DTQ</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_MBX</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_MTX</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_MBF</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_MPF</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_MPL</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_CYH</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX_ALH</td>
<td>Refer to Note 1</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTSZ_MBFTBL</td>
<td>4</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>VTMAX.AreaSize</td>
<td>0x10000000</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>VTKNL_LVL</td>
<td>system.system_IPL</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td></td>
<td>VTIM_LVL</td>
<td>clock.IPL</td>
<td>kernel_id.h</td>
</tr>
<tr>
<td>Classification</td>
<td>Macro</td>
<td>Definition</td>
<td>Where</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Error code</td>
<td>E_SYS</td>
<td>-5</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_NOSPT</td>
<td>-9</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_RSFN</td>
<td>-10</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_RSATR</td>
<td>-11</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_PAR</td>
<td>-17</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_ID</td>
<td>-18</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_CTX</td>
<td>-25</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_MACV</td>
<td>-26</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_OACV</td>
<td>-27</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_ILUSE</td>
<td>-28</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_NOMEM</td>
<td>-33</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_NOID</td>
<td>-34</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_OBJ</td>
<td>-41</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_NOEXS</td>
<td>-42</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_QOVR</td>
<td>-43</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_RLWAI</td>
<td>-49</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_TMOUT</td>
<td>-50</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_DLT</td>
<td>-51</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>E_CLS</td>
<td>-52</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td>EV_RST</td>
<td>-127</td>
<td>itron.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protection</td>
<td>TDOM_SELF</td>
<td>0</td>
<td>kernel.h</td>
</tr>
<tr>
<td>extension</td>
<td>TACP_SHARED</td>
<td>(((1u &lt;&lt; (VTMAX_DOMAIN)) -1)</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TACT_SRW</td>
<td>(TACP_SHARED, TACP_SHARED, TACP_SHARED)</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TACT_SRO</td>
<td>(TACP_SHARED, 0, TACP_SHARED)</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TPM_READ</td>
<td>1</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TPM_WRITE</td>
<td>2</td>
<td>kernel.h</td>
</tr>
<tr>
<td></td>
<td>TPM_EXEC</td>
<td>4</td>
<td>kernel.h</td>
</tr>
</tbody>
</table>

Note 1  Refer to “20.7 Maximum ID (maxdefine)”.  
Note 2  These macros can be describe only at the right of an initial assignment statement.
18.3 Function Macros

The following lists the function macros. The function macros are defined by either of following header files.

- `<ri_root>\inc600\kernel.h`
- `<ri_root>\inc600\itron.h`, which is included by kernel.h

18.3.1 Macros for Error Code

1) ER MERCD ( ER ercd )
   Return the main error code of ercd.

2) ER SERCD ( ER ercd )
   Return sub error code of ercd.

3) ER ERCD ( ER mercd, ER sercd )
   Return the error code from the main error code indicated by mercd and sub error code indicated by sercd.

Note In the error code returned from the RI600PX, all sub error code is -1, and all main error code is same as the value described in Table 18-2.

18.3.2 Macros for Data Queue

1) SIZE TSZ_DTQ ( UINT dtqcnt )
   Returns the size of a data queue area in which the dtqcnt number of data items can be stored. (in bytes)

18.3.3 Macros for Fixed-sized Memory Pool

1) SIZE TSZ_MPF ( UINT blkcnt, UINT blksz )
   Returns the size of a fixed-sized memory pool from which blkcnt number of blksz-byte memory blocks can be acquired. (in bytes)

2) SIZE TSZ_MPFMB ( UINT blkcnt, UINT blksz )
   Returns the size of the management area required for a fixed-sized memory pool from which blkcnt number of blksz-byte memory blocks can be acquired. (in bytes)

18.3.4 Macros for Domain

1) ATR TA_DOM ( ID domid )
   Returns attribute which represents to belong to the domain indicated by domid. This macro is used for bit7-4 of tskatr (task attribute) at task creation.
18.3.5 Access permission

1) ACPTN TACP ( ID domid )
   Returns access permission pattern that represents “only the domain indicated by domid can access”.

2) ACVCT TACT_PRW ( ID domid )
   Returns access permission vector that represents “all types of access (operand-read, operand-write, execution) are permitted only for the domain indicated by domid”.

3) ACVCT TACT_PRO ( ID domid )
   Returns access permission vector that represents “operand-wirte access is not permitted for all domain, operand-read and execution access are permitted only for the domain indicated by domid”.

4) ACVCT TACT_SRPW ( ID domid )
   Returns access permission vector that represents “operand-read and execution access are permitted for all domain, operand-write access is permitted only for the domain indicated by domid”.

CHAPTER 19 SERVICE CALLS

This chapter describes the service calls supported by the RI600PX.

19.1 Outline

The service calls provided by the RI600PX are service routines provided for indirectly manipulating the resources (tasks, semaphores, etc.) managed by the RI600PX from a processing program. The service calls provided by the RI600PX are listed below by management module.

- Task management functions
  - cre_tsk
  - acre_tsk
  - del_tsk
  - act_tsk
  - iact_tsk
  - can_act
  - ican_act
  - sta_tsk
  - ista_tsk
  - ext_tsk
  - exd_tsk
  - ter_tsk
  - chg_pri
  - ichg_pri
  - get_pri
  - iget_pri
  - ref_tsk
  - iref_tsk
  - ref_tst
  - iref_tst

- Task dependent synchronization functions
  - slp_tsk
  - tslp_tsk
  - wup_tsk
  - iwup_tsk
  - can_wup
  - ican_wup
  - rel_wai
  - irel_wai
  - sus_tsk
  - isus_tsk
  - rsm_tsk
  - irsm_tsk
  - frsm_tsk
  - ifrsm_tsk
  - dly_tsk

- Task exception handling functions
  - def_tex
  - ras_tex
  - iras_tex
  - dis_tex
  - ena_tex
  - sns_tex
  - ref_tex
  - iref_tex

- Synchronization and communication functions (semaphores)
  - cre_sem
  - acre_sem
  - del_sem
  - wai_sem
  - pol_sem
  - ipol_sem
  - twai_sem
  - sig_sem
  - isig_sem
  - ref_sem
  - iref_sem

- Synchronization and communication functions (eventflags)
  - cre_flg
  - acre_flg
  - del_flg
  - set_flg
  - iset_flg
  - clr_flg
  - iclr_flg
  - wai_flg
  - pol_flg
  - ipol_flg
  - twai_flg
  - ref_flg

- Synchronization and communication functions (data queues)
  - cre_dtq
  - acre_dtq
  - del_dtq
  - snd_dtq
  - psnd_dtq
  - ipsnd_dtq
  - tsnd_dtq
  - fsnd_dtq
  - ifsnd_dtq
  - rcv_dtq
  - prcv_dtq
  - iprcv_dtq
  - trcv_dtq
  - ref_dtq
  - iref_dtq

- Synchronization and communication functions (mailboxes)
  - cre_mbx
  - acre_mbx
  - del_mbx
  - snd_mbx
  - isnd_mbx
  - rcv_mbx
  - prcv_mbx
  - iprcv_mbx
  - trcv_mbx
  - ref_mbx
  - iref_mbx

- Extended synchronization and communication functions (mutexes)
  - cre_mtx
  - acre_mtx
  - del_mtx
  - loc_mtx
  - ploc_mtx
  - tloc_mtx
  - unl_mtx
  - ref_mtx
- Extended synchronization and communication functions (message buffers)
  cre_mbf    acre_mbf    del_mbf    snd_mbf
  psnd_mbf   ipsnd_mbf   tsnd_mbf   rcv_mbf
  prcv_mbf   trcv_mbf    ref_mbf    iref_mbf

- Memory pool management functions (fixed-sized memory pools)
  cre_mpf    acre_mpf    del_mpf    get_mpf
  pget_mpf   ipget_mpf   tget_mpf   rel_mpf
  irel_mpf   iref_mpf

- Memory pool management functions (variable-sized memory pools)
  cre mpl    acre mpl    del mpl    get mpl
  pg get mpl  ip get mpl  tget mpl   rel mpl
  irel mpl   iref mpl

- Time management functions
  set_tim    iset_tim   get_tim    iget_tim
  cre Cyc    acre Cyc    del Cyc    sta Cyc
  ista Cyc   istp Cyc   stp Cyc    istp Cyc
  iref Cyc   cre alm    acre alm    del alm
  sta alm    ista alm   stp alm    istp alm
  ref alm    iref alm

- System state management functions
  rot Rdq    irot Rdq    get Tid    iget Tid
  loc cpu    iloc cpu    unl cpu    iunl cpu
  dis disp   ena disp    sns ctx    sns loc
  sns disp   sns dpn    vsys dwn    ivsys dwn
  vsta knl   ivsta knl

- Interrupt management functions
  chg ims    ichg ims    get ims    iget ims

- System configuration management functions
  ref Ver    iref Ver

- Object reset functions
  vrst dtq    vrst mbx    vrst mbf    vrst mpf

- Memory object management functions
  ata mem    det mem    sac mem    vprb mem
  ref mem

19.1.1 Method for calling service calls
The service calls can be calls by the same way as normal C-language function.

Note To call the service calls provided by the RI600PX from a processing program, the header files listed below must be coded (include processing).

  kernel.h: Standard header file
  kernel_id.h System information header file, which is generated by the cfg600px
19.2 Explanation of Service Call

The following explains the service calls supported by the RI600PX, in the format shown below.

1) 

2) Outline

3) C format

4) Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5) Explanation

6) Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 ) Name
Indicates the name of the service call.

2 ) Outline
Outlines the functions of the service call.

3 ) C format
Indicates the format to be used when describing a service call to be issued in C language.

4 ) Parameter(s)
Service call parameters are explained in the following format.

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

A ) Parameter classification
I: Parameter input to RI600PX.
O: Parameter output from RI600PX.

B ) Parameter data type
C ) Description of parameter

5 ) Explanation
Explains the function of a service call.

6 ) Return value
Indicates a service call's return value using a macro and value.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

A ) Macro of return value
B ) Value of return value
C ) Description of return value
19.2.1 Task management functions

The following shows the service calls provided by the RI600PX as the task management functions.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_tsk</td>
<td>Create task</td>
<td>Task</td>
</tr>
<tr>
<td>acre_tsk</td>
<td>Create task (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_tsk</td>
<td>Delete task</td>
<td>Task</td>
</tr>
<tr>
<td>act_tsk</td>
<td>Activate task (queues an activation request)</td>
<td>Task</td>
</tr>
<tr>
<td>iact_tsk</td>
<td>Activate task (queues an activation request)</td>
<td>Non-task</td>
</tr>
<tr>
<td>can_act</td>
<td>Cancel task activation requests</td>
<td>Task</td>
</tr>
<tr>
<td>ican_act</td>
<td>Cancel task activation requests</td>
<td>Non-task</td>
</tr>
<tr>
<td>sta_tsk</td>
<td>Activate task (does not queue an activation request)</td>
<td>Task</td>
</tr>
<tr>
<td>ista_tsk</td>
<td>Activate task (does not queue an activation request)</td>
<td>Non-task</td>
</tr>
<tr>
<td>ext_tsk</td>
<td>Terminate invoking task</td>
<td>Task</td>
</tr>
<tr>
<td>exd_tsk</td>
<td>Terminate and delete invoking task</td>
<td>Task</td>
</tr>
<tr>
<td>ter_tsk</td>
<td>Terminate task</td>
<td>Task</td>
</tr>
<tr>
<td>chg_pri</td>
<td>Change task priority</td>
<td>Task</td>
</tr>
<tr>
<td>ichg_pri</td>
<td>Change task priority</td>
<td>Non-task</td>
</tr>
<tr>
<td>get_pri</td>
<td>Reference task current priority</td>
<td>Task</td>
</tr>
<tr>
<td>iget_pri</td>
<td>Reference task current priority</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_tsk</td>
<td>Reference task state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_tsk</td>
<td>Reference task state</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_tst</td>
<td>Reference task state (simplified version)</td>
<td>Task</td>
</tr>
<tr>
<td>iref_tst</td>
<td>Reference task state (simplified version)</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline
Create task.

C format

```c
ER      cre_tsk (ID tskid, T_CTSK *pk_ctsk );
ER_ID   acre_tsk ( T_CTSK *pk_ctsk );
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>tskid; ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>T_CTSK *pk_ctsk</td>
<td>Pointer to the packet containing the task creation information.</td>
</tr>
</tbody>
</table>

[Task creation information packet : T_CTSK]

```c
type struct  t_ctsk {
   ATR     tskatr;         /*Task attribute*/
   VP_INT  exinf;          /*Extended information*/
   FP      task;           /*Task start address*/
   PRI     itskpri;        /*Task initial priority*/
   SIZE    stksz;          /*User stack size (in bytes)*/
   VP      stk;            /*Start address of user stack*/
} T_CTSK;
```

Explanation

This service call can be called from tasks that belong to Trusted Domain. The cre_tsk creates a task with task ID indicated by tskid according to the content of pk_ctsk. The acre_tsk creates a task according to the content of pk_ctsk, and returns the created task ID. The processing performed at task creation is shown in Table 19-2.

Table 19-2 Processing Performed at Task Creation

<table>
<thead>
<tr>
<th>No.</th>
<th>Content of processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clears the number of queued activation requests.</td>
</tr>
<tr>
<td>2</td>
<td>Resets the task state so that the task exception handling routine is not defined.</td>
</tr>
</tbody>
</table>

1) Task attribute (tskatr)
The following are specified for tskatr:

```c
tskatr := { TA_HLNG | [TA_ACT] | [TA_DOM (domid)] }  
```
The bit position of tskat is shown as follows.

<table>
<thead>
<tr>
<th>bit15 ~ bit8</th>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Domain ID 'TA_DOM(domid)'</td>
<td>0</td>
<td>0</td>
<td>TA_ACT :1</td>
<td>TA_HLNG : 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TA_HLNG** (= 0x0000)
  Only C-language is supported for task description language.

- **TA_ACT** (= 0x0002)
  When the TA_ACT attribute is specified, the created task makes a transition to the READY state. The processing performed at task activation is shown in Table 19-3. When the TA_ACT attribute is not specified, the created task makes a transition to the DORMANT state.

- **TA_DOM(domid)**
  The created task belong to the domain indicated by domid. When 0 is specified for domid or TA_DOM(domid) is not specified, the created task belongs to the domain that the invoking task belong to.

  **Note** For detail of TA_DOM macro, refer to "18.3.4 Macros for Domain".

2) **Extended information (exinf)**

When the task is activated by TA_ACT attribute, act_tsk or iact_tsk, exinf is passed to the task as argument. And exinf is passed to the task exception handling routine. The exinf can be widely used by the user, for example, to set information concerning the task.

3) **Task start address (task)**

Specify the task start address for task.

4) **Task initial priority (itskpri)**

Specify initial priority of the task for itskpri. Ranges of the value that can be specified are from 1 to TMAX_TPRI.

5) **User stack size (stksz), Start address of user stack (stk)**

The application acquires user stack area, and specifies the start address for stk and the size for stksz. The user stack area must satisfy the following.

A) The stk must be 16-bytes boundary. If not, error E_PAR is returned.

B) The stksz must be multiple of 16. If not, error E_PAR is returned.

C) The user stack area must not overlap with either all user stacks and all memory objects. If not, an error is not detected and correct system operation cannot be guaranteed.

  **Note** The μITRON4.0 specification defines the function that the kernel allocates user stack area when NULL is specified for stk. But RI600PX does not support this function.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_tsk. (Created task ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_tsk.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either bit0, bit2, bit3 or bit8-15 in tskatr is 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- VTMAX_DOMAIN &lt; (Value of bit4-7 of tskatr)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_ctsk == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- task == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- itskpri &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TMAX_TPRI &lt; itskpri</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stk is not 16-bytes boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stksz is not multiple of 16.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stksz &lt; (lower bound value described in Table 20-8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- VTMAX_AREASIZE &lt; stksz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stk + stksz &gt; 0x100000000</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_tsk)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pk_ctsk has not been</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOMEM</td>
<td>-33</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- stk == NULL</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_tsk)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_tsk)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid exists.</td>
</tr>
</tbody>
</table>
**del_tsk**

### Outline
Delete task.

### C format
```
ER del_tsk (ID tskid);
```

### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

### Explanation
This service call can be called from tasks that belong to Trusted Domain.
This service call deletes the task indicated by tskid.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is not in the DORMANT state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Activate task (queues an activation request).

C format
ER      act_tsk (ID tskid);
ER      iact_tsk (ID tskid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>tskid;</td>
</tr>
</tbody>
</table>

ID number of the task.

TSK_SELF: Invoking task.
Value: ID number of the task.

Explanation
These service calls move the task specified by parameter tskid from the DORMANT state to the READY state. As a result, the target task is queued at the end on the ready queue corresponding to the initial priority and becomes subject to scheduling by the RI600PX. At this time, the following processing is done.

Table 19-3  Processing Performed at Task Activation

<table>
<thead>
<tr>
<th>No.</th>
<th>Content of processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initializes the task’s base priority and current priority.</td>
</tr>
<tr>
<td>2</td>
<td>Clears the number of queued wake-up requests.</td>
</tr>
<tr>
<td>3</td>
<td>Clears the number of nested suspension count.</td>
</tr>
<tr>
<td>4</td>
<td>Clears pending exception code.</td>
</tr>
<tr>
<td>5</td>
<td>Disables task exception</td>
</tr>
</tbody>
</table>

If the target task has been moved to a state other than the DORMANT state when this service call is issued, this service call does not move the state but increments the activation request counter (by added 1 to the activation request counter).

Note 1  The activation request counter managed by the RI600PX is configured in 8-bit widths. If the number of activation requests exceeds the maximum count value 255 as a result of issuing this service call, the counter manipulation processing is therefore not performed but "E_QOVR" is returned.

Note 2  Extended information specified at creating the task is passed to the task activated by issuing these service calls.
### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_ID   | -18   | Invalid ID number.  
- $tskid < 0$  
- $tskid > VTMAX_TSK$  
- When $iact_tsk$ was issued from a non-task, $TSK_SELF$ was specified for $tskid$. |
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- The $iact_tsk$ was issued from task.  
- The $act_tsk$ was issued from non-task.  
- This service call was issued in the status "PSW.IPL > kernel interrupt mask level". |
| E_MACV | -26   | Memory access violation. (only for $act_tsk$)  
- Stack pointer points out of user stack for invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The task specified by $tskid$ does not exist. |
| E_QOVR | -43   | Queuing overflow.  
- Activation request count exceeded 255. |
can_act
ican_act

Outline
Cancel task activation requests.

C format
ER_UINT can_act (ID tskid);
ER_UINT ican_act (ID tskid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>TSK_SELF: Invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: ID number of the task.</td>
</tr>
</tbody>
</table>

Explanation
This service call cancels all of the activation requests queued to the task specified by parameter tskid (sets the activation request counter to 0).
When this service call is terminated normally, the number of cancelled activation requests is returned.

Note  This service call does not perform status manipulation processing but performs the setting of activation request counter. Therefore, the task does not move from a state such as the READY state to the DORMANT state.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion (activation request count).</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Activation request count is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is in the DORMANT state.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When the iact_tsk was issued from a non-task, TSK_SELF was specified for tskid.</td>
</tr>
</tbody>
</table>
Context error.
- This service call was issued in the CPU locked state.
- This service call was issued in the status "PSW.IPL > kernel interrupt mask level".

Note When the ican_act is issued from task or the can_act is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.

Non-existent object.
- The task specified by tskid does not exist.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
sta_tsk
ista_tsk

Outline
Activate task (does not queue an activation request).

C format
ER  sta_tsk (ID tskid, VP_INT stacd);
ER  ista_tsk (ID tskid, VP_INT stacd);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>VP_INT</td>
<td>Start code of the task.</td>
</tr>
</tbody>
</table>

Explanation
These service calls move the task specified by parameter tskid from the DORMANT state to the READY state. As a result, the target task is queued at the end on the ready queue corresponding to the initial priority and becomes subject to scheduling by the RI600PX. At this time, processing described in Table 19-3 is done. These service calls do not perform queuing of activation requests. If the target task is in a state other than the DORMANT state, the status manipulation processing for the target task is therefore not performed but “E_OBJ” is returned. The stacd is passed to the target task.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The ista_tsk was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The sta_tsk was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for sta_tsk)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is not in the DORMANT state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
**ext_tsk**

**Outline**
Terminates an invoking task.

**C format**
```c
void ext_tsk (void);
```

**Parameter(s)**
None.

**Explanation**
This service call moves the invoking task from the RUNNING state to the DORMANT state. As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. At this time, the following processing is done.

### Table 19-4 Processing Performed at Task Termination

<table>
<thead>
<tr>
<th>No.</th>
<th>Content of processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unlocks the mutexes which are locked by the terminated task. (processing equivalent to <code>unl_mtx</code> will be executed)</td>
</tr>
</tbody>
</table>

The CPU locked state and dispatching disabled state is cancelled. If an activation request has been queued to the invoking task (the activation request counter > 0) when this service call is issued, this service call moves the task from the RUNNING state to the DORMANT state, decrements the activation request counter (by subtracting 1 from the activation request counter), and then moves the task from the DORMANT state to the READY state. At this time, processing described in Table 19-3 is done.

This service call does not return. In the following cases, this service call causes **SYSTEM DOWN**.

- This service call was issued from non-task.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”

**Note 1** When the return instruction is issued in the task entry function, the same processing as `ext_tsk` is performed.

**Note 2** This service call does not have the function to automatically free the resources except the mutex hitherto occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before it terminates.

**Return value**
None.


**Outline**
Terminate and delete invoking task.

**C format**
```c
void    exd_tsk (void);
```

**Parameter(s)**
None.

**Explanation**
This service call terminates the invoking task normally and deletes the task.
The processing performed at task termination is shown in Table 19-4.
The CPU locked state and dispatching disabled state is cancelled.
This service call does not return. In the following cases, this service call causes SYSTEM DOWN.

- This service call was issued from non-task.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”

**Note**
This service call does not have the function to automatically free the resources except the mutex hitherto occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before it terminates.

**Return value**
None.
Outline
Terminates a task.

C format

```c
ER ter_tsk (ID tskid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

Explanation

This service call forcibly moves the task specified by parameter tskid to the DORMANT state.
As a result, the target task is excluded from the RI600PX scheduling subject.
At this time, processing described in Table 19-4 is done.

If an activation request has been queued to the target task (the activation request counter > 0) when this service call is
issued, this service call moves the task to the DORMANT state, decrements the activation request counter (by subtracting
1 from the activation request counter), and then moves the task from the DORMANT state to the READY state. At this
time, processing described in Table 19-3 is done.

Note: This service call does not have the function to automatically free the resources except the mutex hitherto
occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before
it terminates.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal service call use.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is the invoking task.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is in the DORMANT state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
Outline

Change task priority.

C format

ER      chg_pri (ID tskid, PRI tskpri);
ER      ichg_pri (ID tskid, PRI tskpri);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TSK_SELF: Invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>PRI tskpri;</td>
<td>New base priority of the task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPRI_INI: Initial priority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: New base priority of the task.</td>
</tr>
</tbody>
</table>

Explanation

This service call changes the base priority of the task specified by parameter tskid to a value specified by parameter tskpri.

The changed base priority is effective until the task terminates or this service call is issued. When next the task is activated, the base priority is the initial priority which is specified at the task creation.

This service call also changes the current priority of the target task to a value specified by parameter tskpri. However, the current priority is not changed when the target task has locked mutexes.

If the target task has locked mutexes or is waiting for mutex to be locked and if tskpri is higher than the ceiling priority of either of the mutexes, this service call returns "E_ILUSE".

When the current priority is changed, the following state variations are generated.

1) When the target task is in the RUNNING or READY state.
   This service call re-queues the task at the end of the ready queue corresponding to the priority specified by parameter tskpri.

2) When the target task is queued to a wait queue of the object with TA_TPRI or TA_CEILING attribute.
   This service call re-queues the task to the wait queue corresponding to the priority specified by parameter tskpri.
   When two or more tasks of same current priority as tskpri, this service call re-queues the target task at the end among their tasks.

Example

When three tasks (task A: priority level 10, task B: priority level 11, task C: priority level 12) are queued to the semaphore wait queue in the order of priority, and the priority level of task B is changed from 11 to 9, the wait order will be changed as follows.
Note For current priority and base priority, refer to "8.2.2 Current priority and base priority".

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error. &lt;br&gt; - tskpri &lt; 0 &lt;br&gt; - tskpri &gt; TMAX_TPRI</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. &lt;br&gt; - tskid &lt; 0 &lt;br&gt; - tskid &gt; VTMAX_TSK &lt;br&gt; - When ichg_pri was issued from a non-task, TSK_SELF was specified for tskid.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error. &lt;br&gt; - This service call was issued in the CPU locked state. &lt;br&gt; - The ichg_pri was issued from task. &lt;br&gt; - The chg_pri was issued from non-task. &lt;br&gt; - This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for chg_pri) &lt;br&gt; - Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call. &lt;br&gt; - tskpri &lt; The ceiling priority of the mutex locked by the target task. &lt;br&gt; - tskpri &lt; The ceiling priority of the mutex by which the target task waits for lock.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. &lt;br&gt; - Specified task is in the DORMANT state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object. &lt;br&gt; - The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
**get_pri**

**iget_pri**

**Outline**
Reference task current priority.

**C format**

```c
ER get_pri (ID tskid, PRI *p_tskpri);
ER iget_pri (ID tskid, PRI *p_tskpri);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task. &lt;br&gt;<strong>TSK_SELF:</strong> Invoking task. &lt;br&gt;<strong>Value:</strong> ID number of the task.</td>
</tr>
<tr>
<td>O</td>
<td>PRI *p_tskpri;</td>
<td>Pointer to the area returning the current priority of the task.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call stores the current priority of the task specified by parameter `tskid` in the area specified by parameter `p_tskpri`.

**Note**  For current priority and base priority, refer to “8.2.2 Current priority and base priority”.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error. &lt;br&gt; - <code>p_tskpri</code> == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. &lt;br&gt; - <code>tskid</code> &lt; 0 &lt;br&gt; - <code>tskid</code> &gt; VMAX_TSK &lt;br&gt; - When this service call was issued from a non-task, <strong>TSK_SELF</strong> was specified for <code>tskid</code>.</td>
</tr>
</tbody>
</table>
Context error.
- This service call was issued in the CPU locked state.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.

Note When the iget_pri is issued from task or the get_pri is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.

Memory access violation. (only for get_pri)
- The operand-write access to the area indicated by p_tskpri has not been permitted to the invoking task.

Object state error.
- Specified task is in the DORMANT state.

Non-existent object.
- The task specified by tskid does not exist.
ref_tsk
iref_tsk

Outline
Reference task state.

C format
ER ref_tsk (ID tskid, T_RTSK *pk_rtsk);
ER iref_tsk (ID tskid, T_RTSK *pk_rtsk);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TSK_SELF: Invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: ID number of the task.</td>
</tr>
<tr>
<td>O</td>
<td>T_RTSK *pk_rtsk;</td>
<td>Pointer to the packet returning the task state.</td>
</tr>
</tbody>
</table>

(Task state packet: T_RTSK)

typedef struct t_rtsk {
    STAT tskstat;        /*Current state*/
    PRI tskpri;         /*Current priority*/
    PRI tskbpri;        /*Base priority*/
    STAT tskwait;        /*Reason for waiting*/
    ID wobjid;         /*Object ID number for which the task is waiting*/
    TMO lefttmo;        /*Remaining time until time-out*/
    UINT actcnt;        /*Activation request count*/
    UINT wupcnt;        /*Wake-up request count*/
    UINT suscnt;        /*Suspension count*/
} T_RTSK;

Explanation
Stores task state packet (current state, current priority, etc.) of the task specified by parameter tskid in the area specified by parameter pk_rtsk.

- **tskstat**
  Stores the current state.
  - **TTS_RUN**: RUNNING state
  - **TTS_RDY**: READY state
  - **TTS_WAI**: WAITING state
  - **TTS_SUS**: SUSPENDED state
  - **TTS_WAS**: WAITING-SUSPENDED state
  - **TTS_DMT**: DORMANT state
- tskpri
Stores the current priority.
The tskpri is effective only when the tskstat is other than TTS_DMT.

- tskbpri
Stores the base priority.
The tskbpri is effective only when the tskstat is other than TTS_DMT.

- tskwait
Stores the reason for waiting.
The tskwait is effective only when the tskstat is TTS_WAI or TTS_WAS.

  TTW_SLP: Sleeping state caused by slp_tsk or tslp_tsk
  TTW_DLY: Delayed state caused by dly_tsk
  TTW_SEM: WAITING state for a semaphore resource caused by wai_sem or twai_sem
  TTW_FLG: WAITING state for an eventflag caused by wai_flg or twai_flg
  TTW_SDTQ: Sending WAITING state for a data queue caused by snd_dtq or tsnd_dtq
  TTW_RDTQ: Receiving WAITING state for a data queue caused by rcv_dtq or trcv_dtq
  TTW_MBX: Receiving WAITING state for a mailbox caused by rcv_mbx or trcv_mbx
  TTW_MTX: WAITING state for a mutex caused by loc_mtx or tloc_mtx
  TTW_SMBF: Sending WAITING state for a message buffer caused by snd_mbf or tsnd_mbf
  TTW_RMBF: Receiving WAITING state for a message buffer caused by rcv_mbf or trcv_mbf
  TTW_MPF: WAITING state for a fixed-sized memory block caused by get_mpf or tget_mpf
  TTW_MPL: WAITING state for a variable-sized memory block caused by get_mpl or tget_mpl

- wobjid
Stores the object (such as semaphore, eventflag, etc.) ID number for which the task waiting.
The wobjid is effective only when the tskwait is TTW_SEM or TTW_FLG or TTW_SDTQ or TTW_RDTQ or TTW_MBX or TTW_MTX or TTW_SMBF or TTW_RMBF or TTW_MPF or TTW_MPL.

- lefttmo
Stores the remaining time until time-out (in millisecond).
The TMO_FEVR is stored for waiting forever.
The lefttmo is effective only when the tskstat is TTS_WAI or TTS_WAS, and the tskwait is other than TTW_DLY.

Note The lefttmo is undefined when the tskwait is TTW_DLY.

- actcnt
Stores the activation request count.

- wupcnt
Stores the wake-up request count.
The wupcnt is effective only when the tskstat is other than TTS_DMT.

- suscnt
Stores the suspension count.
The suscnt is effective only when the tskstat is other than TTS_DMT.
Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rtsk == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When this service call was issued from a non-task, TSK_SELF was specified for tsid.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_tsk is issued from task or the ref_tsk is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_tsk)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rtsk has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tsid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Reference task state (simplified version).

C format

ER  ref_tst (ID tskid, T_RTST *pk_rtst);
ER  iref_tst (ID tskid, T_RTST *pk_rtst);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td></td>
<td>tskid;</td>
<td>Value: ID number of the task. TSK_SELF: Invoking task. TTS_RUN: RUNNING state TTS_RDY: READY state TTS_WAI: WAITING state TTS_SUS: SUSPENDED state TTS_WAS: WAITING-SUSPENDED state TTS_DMT: DORMANT state</td>
</tr>
<tr>
<td>O</td>
<td>T_RTST</td>
<td>Pointer to the packet returning the task state.</td>
</tr>
<tr>
<td></td>
<td>*pk_rtst;</td>
<td></td>
</tr>
</tbody>
</table>

I/O Parameter Description

I ID tskid; ID number of the task.

O T_RTST *pk_rtst; Pointer to the packet returning the task state.

typedef struct t_rtst {
    STAT tskstat; /*Current state*/
    STAT tskwait; /*Reason for waiting*/
} T_RTST;

Explanation
Stores task state packet (current state, reason for waiting) of the task specified by parameter tskid in the area specified by parameter pk_rtst.
Used for referencing only the current state and reason for wait among task information.
Response becomes faster than using ref_tsk or iref_tsk because only a few information items are acquired.

- tskstat
Stores the current state.

    TTS_RUN: RUNNING state
    TTS_RDY: READY state
    TTS_WAI: WAITING state
    TTS_SUS: SUSPENDED state
    TTS_WAS: WAITING-SUSPENDED state
    TTS_DMT: DORMANT state

- tskwait
Stores the reason for waiting.
The tskwait is effective only when the tskstat is TTS_WAI or TTS_WAS.

    TTW_SLP: Sleeping state caused by slip_tsk or tslp_tsk
    TTW_DLY: Delayed state caused by dly_tsk
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( pk_rtst ) \text{\textasciitilde} NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( tsid &lt; 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( tsid ) \text{\textgreater} VMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When this service call was issued from a non-task, \text{TSK_SELF} was specified for \text{tskid}.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL \text{\textgreater} kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_tst is issued from task or the ref_tst is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_tst)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by ( pk_rtst ) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by \text{tskid} does not exist.</td>
</tr>
</tbody>
</table>
19.2.2 Task dependent synchronization functions

The following shows the service calls provided by the RI600PX as the task dependent synchronization functions.

Table 19-5 Task Dependent Synchronization Functions

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>slp_tsk</td>
<td>Put task to sleep (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>tslp_tsk</td>
<td>Put task to sleep (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>wup_tsk</td>
<td>Wake-up task</td>
<td>Task</td>
</tr>
<tr>
<td>iwup_tsk</td>
<td>Wake-up task</td>
<td>Non-task</td>
</tr>
<tr>
<td>can_wup</td>
<td>Cancel task wake-up requests</td>
<td>Task</td>
</tr>
<tr>
<td>ican_wup</td>
<td>Cancel task wake-up requests</td>
<td>Non-task</td>
</tr>
<tr>
<td>rel_wai</td>
<td>Release task from waiting</td>
<td>Task</td>
</tr>
<tr>
<td>irel_wai</td>
<td>Release task from waiting</td>
<td>Non-task</td>
</tr>
<tr>
<td>sus_tsk</td>
<td>Suspend task</td>
<td>Task</td>
</tr>
<tr>
<td>isus_tsk</td>
<td>Suspend task</td>
<td>Non-task</td>
</tr>
<tr>
<td>rsm_tsk</td>
<td>Resume suspended task</td>
<td>Task</td>
</tr>
<tr>
<td>irsm_tsk</td>
<td>Resume suspended task</td>
<td>Non-task</td>
</tr>
<tr>
<td>frsm_tsk</td>
<td>Forcibly resume suspended task</td>
<td>Task</td>
</tr>
<tr>
<td>ifrsm_tsk</td>
<td>Forcibly resume suspended task</td>
<td>Non-task</td>
</tr>
<tr>
<td>dly_tsk</td>
<td>Delay task</td>
<td>Task</td>
</tr>
</tbody>
</table>
**slp_tsk**

**Outline**
Put task to sleep (waiting forever).

**C format**
```c
ER slp_tsk (void);
```

**Parameter(s)**
None.

**Explanation**
As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. If a wake-up request has been queued to the target task (the wake-up request counter > 0) when this service call is issued, this service call does not move the state but decrements the wake-up request counter (by subtracting 1 from the wake-up request counter). The sleeping state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sleeping State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A wake-up request was issued as a result of issuing <strong>wup_tsk</strong>.</td>
<td><strong>E_OK</strong></td>
</tr>
<tr>
<td>A wake-up request was issued as a result of issuing <strong>iwup_tsk</strong>.</td>
<td><strong>E_OK</strong></td>
</tr>
<tr>
<td>Forced release from waiting (accept <strong>rel_wai</strong> while waiting).</td>
<td><strong>E_RLWAI</strong></td>
</tr>
<tr>
<td>Forced release from waiting (accept <strong>irel_wai</strong> while waiting).</td>
<td><strong>E_RLWAI</strong></td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E_OK</strong></td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| **E_CTX**| -25   | Context error.  
- This service call was issued from a non-task.  
- This service call was issued in the CPU locked state.  
- This service call was issued in the dispatching disabled state.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.
| **E_MACV**| -26   | Memory access violation.  
- Stack pointer points out of user stack for invoking task. |
| **E_RLWAI**| -49  | Forced release from the WAITING state.  
- Accept **rel_wai**/**irel_wai** while waiting. |
Outline
Put task to sleep (with time-out).

C format

ER tslp_tsk (TMO tmout);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_FEVR: Waiting forever.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_POL: Polling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: Specified time-out.</td>
</tr>
</tbody>
</table>

Explanation
This service call moves the invoking task from the RUNNING state to the WAITING state (sleeping state). As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. If a wake-up request has been queued to the target task (the wake-up request counter > 0) when this service call is issued, this service call does not move the state but decrements the wake-up request counter (by subtracting 1 from the wake-up request counter). The sleeping state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sleeping State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A wake-up request was issued as a result of issuing wup_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A wake-up request was issued as a result of issuing iwup_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
</tbody>
</table>

Note  When TMO_FEVR is specified for wait time tmout, processing equivalent to slp_tsk will be executed.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tmout &lt; -1</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept <code>rel_wai</code>/<code>irel_wai</code> while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
</tbody>
</table>
**wup_tsk**  
**iwup_tsk**

**Outline**

Wake-up task.

**C format**

```c
ER      wup_tsk (ID tskid);
ER      iwup_tsk (ID tskid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

**Explanation**

These service calls cancel the WAITING state (sleeping state) of the task specified by parameter `tskid`. As a result, the target task is moved from the sleeping state to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

If the target task is in a state other than the sleeping state when this service call is issued, this service call does not move the state but increments the wake-up request counter (by added 1 to the wake-up request counter).

**Note** The wake-up request counter managed by the RI600PX is configured in 8-bit widths. If the number of wake-up requests exceeds the maximum count value 255 as a result of issuing this service call, the counter manipulation processing is therefore not performed but "E_QOVR" is returned.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid &lt; 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid &gt; VTMAX_TSK</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When iwup_tsk was issued from a non-task, TSK_SELF was specified for tskid.</td>
</tr>
</tbody>
</table>
### Macro Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_CTX | -25   | Context error.  
  - This service call was issued in the CPU locked state.  
  - The iwup_tsk was issued from task.  
  - The wup_tsk was issued from non-task.  
  - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV| -26   | Memory access violation. (only for wup_tsk)  
  - Stack pointer points out of user stack for invoking task. |
| E_OBJ | -41   | Object state error.  
  - Specified task is in the DORMANT state. |
| E_NOEXS| -42   | Non-existent object.  
  - The task specified by `tskid` does not exist. |
| E_QOVR| -43   | Queuing overflow.  
  - Wake-up request count exceeded 255. |
### can_wup
### ican_wup

#### Outline
Cancel task wake-up requests.

#### C format
```c
ER_UINT can_wup (ID tskid);
ER_UINT ican_wup (ID tskid);
```

#### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

- **TSK_SELF**: Invoking task.
- **Value**: ID number of the task.

#### Explanation
These service calls cancel all of the wake-up requests queued to the task specified by parameter `tskid` (the wake-up request counter is set to 0), and return the number of cancelled wake-up requests.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0 or more</td>
<td>Normal completion (wake-up request count).</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid</code> &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid</code> &gt; VTMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When this service call was issued from a non-task, TSK_SELF was specified for <code>tskid</code>.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong>: When the ican_wup is issued from task or the can_wup is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is in the DORMANT state.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Release task from waiting.

C format

```c
ER rel_wai (ID tskid);
ER irel_wai (ID tskid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

Explanation

These service calls forcibly cancel the WAITING state of the task specified by parameter tskid. As a result, the target task unlinked from the wait queue and is moved from the WAITING state to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. “E_RLWAI” is returned from the service call that triggered the move to the WAITING state (slp_tsk, wai_sem, or the like) to the task whose WAITING state is cancelled by this service call.

Note 1 These service calls do not perform queuing of forced cancelation requests. If the target task is neither in the WAITING state nor WAITING-SUSPENDED state, “E_OBJ” is returned.

Note 2 The SUSPENDED state is not cancelled by these service calls.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The irel_wai was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The rel_wai was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for rel_wai)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is neither in the WAITING state nor WAITING-SUSPENDED state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
sus_tsk
isus_tsk

Outline
Suspend task.

C format
ER      sus_tsk (ID tskid);
ER      isus_tsk (ID tskid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>tskid;</td>
</tr>
</tbody>
</table>

Parameter(s) Explanation

These service calls move the task specified by parameter tskid from the RUNNING state to the SUSPENDED state, from the READY state to the SUSPENDED state, or from the WAITING state to the WAITING-SUSPENDED state.

If the target task has moved to the SUSPENDED or WAITING-SUSPENDED state when this service call is issued, these service calls return “E_QOVR”.

Note In the RI600PX, the suspend request can not be nested.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When this service call was issued from a non-task, TSK_SELF was specified for tskid.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The isus_tsk was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The sus_tsk was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task is specified in the dispatching disabled state.</td>
</tr>
</tbody>
</table>
## Macro Value Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_MACV | -26 | Memory access violation. (only for sus_tsk)  
- Stack pointer points out of user stack for invoking task. |
| E_OBJ | -41 | Object state error.  
- Specified task is in the DORMANT state.  
- Specified task is in the RUNNING state when isus_tsk is issued in the dispatching disabled state. |
| E_NOEXS | -42 | Non-existent object.  
- The task specified by tskid does not exist. |
| E_QOVR | -43 | Queuing overflow.  
- Specified task is neither in the SUSPENDED state nor WAITING-SUSPENDED state. |
Resume suspended task.

Outline

C format

```c
ER  rsm_tsk (ID tskid);
ER  irsm_tsk (ID tskid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

Explanation

These service calls move the task specified by parameter `tskid` from the SUSPENDED state to the READY state, or from the WAITING-SUSPENDED state to the WAITING state.

**Note 1** These service calls do not perform queuing of forced cancelation requests. If the target task is neither in the SUSPENDED state nor WAITING-SUSPENDED state, “E_OBJ” is returned.

**Note 2** The RI600PX does not support queuing of suspend request. The behavior of the `frsm_tsk` and `ifrsm_tsk`, that can release from the SUSPENDED state even if suspend request has been queued, are same as `rsm_tsk` and `irsm_tsk`.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_ID  | -18   | Invalid ID number.  
- tskid ≤ 0  
- tskid > VMAX_TSK |
| E_CTX | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- The irsm_tsk was issued from task.  
- The rsm_tsk was issued from non-task.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV| -26   | Memory access violation. (only for rsm_tsk)  
- Stack pointer points out of user stack for invoking task. |
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_OBJ      | -41   | **Object state error.**  
- Specified task is neither in the SUSPENDED state nor WAITING-SUSPENDED state. |
| E_NOEXS    | -42   | **Non-existent object.**  
- The task specified by $tskid$ does not exist. |
frsm_tsk
ifrsm_tsk

Outline
Forcibly resume suspended task.

C format
ER  frsm_tsk (ID tskid);
ER  ifrsm_tsk (ID tskid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid</td>
<td>ID number of the task.</td>
</tr>
</tbody>
</table>

Explanation
These service calls cancel all of the suspend requests issued for the task specified by parameter tskid (by setting the suspend request counter to 0). As a result, the target task moves from the SUSPENDED state to the READY state, or from the WAITING-SUSPENDED state to the WAITING state.

Note 1 These service calls do not perform queuing of forced cancelation requests. If the target task is neither in the SUSPENDED state nor WAITING-SUSPENDED state, "E_OBJ" is returned.

Note 2 The RI600PX does not support queuing of suspend request. Therefore, the behavior of these service calls are same as rsm_tsk and irsm_tsk.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VMAX_TSK</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The ifrsm_tsk was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The frsm_tsk was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for frsm_tsk)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specified task is neither in the SUSPENDED state nor WAITING-SUSPENDED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>state.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
**dly_tsk**

**Outline**

Delay task.

**C format**

```c
ER     dly_tsk (RELTIM dlytim);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>RELTIM dlytim;</td>
<td>Amount of time to delay the invoking task (in millisecond).</td>
</tr>
</tbody>
</table>

**Explanation**

This service call moves the invoking task from the RUNNING state to the WAITING state (delayed state). As a result, the invoking task is unlinked from the ready queue and excluded from the RI600PX scheduling subject. The delayed state is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Delayed State Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay time specified by parameter <code>dlytim</code> has elapsed.</td>
<td><code>E_OK</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td><code>E_RLWAI</code></td>
</tr>
</tbody>
</table>

**Note** When 0 is specified as `dlytim`, the delay time is up to next base clock interrupt generation.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>E_OK</code></td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td><code>E_PAR</code></td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dlytim &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</code></td>
</tr>
<tr>
<td><code>E_CTX</code></td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| E_MACV  | -26   | Memory access violation.  
- Stack pointer points out of user stack for invoking task. |
| E_RLWAI | -49   | Forced release from the WAITING state.  
- Accept rel_wai/irel_wai while waiting. |
### 19.2.3 Task exception handling functions

The following shows the service calls provided by the RI600PX as the task exception handling functions.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>def_tex</td>
<td>Define task exception handling routine</td>
<td>Task</td>
</tr>
<tr>
<td>ras_tex</td>
<td>Raise task exception</td>
<td>Task</td>
</tr>
<tr>
<td>iras_tex</td>
<td>Raise task exception</td>
<td>Non-task</td>
</tr>
<tr>
<td>dis_tex</td>
<td>Disable task exception</td>
<td>Task</td>
</tr>
<tr>
<td>ena_tex</td>
<td>Enable task exception</td>
<td>Task</td>
</tr>
<tr>
<td>sns_tex</td>
<td>Reference task exception disabled state</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>ref_tex</td>
<td>Reference task exception state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_tex</td>
<td>Reference task exception state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>

Table 19-6  Task Exception Handling Functions
def_tex

Outline
Define task exception handling routine.

C format
ER  def_tex (ID tskid, T_DTEX *pk_dtex);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task. TSK_SELF: Invoking task. Value: ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>T_DTEX *pk_dtex;</td>
<td>NULL: Cancel the definition of task exception handling routine. Other than NULL: Pointer to the packet containing the task exception handling routine definition information.</td>
</tr>
</tbody>
</table>

[Task exception handling routine definition information packet: T_DTEX]

typedef struct t_dtex {
  ATR texatr;       /*Task exception handling routine attribute*/
  FP texrtn;        /*Task exception handling routine start address*/
} T_DTEX;

Explanation
This service call can be called from tasks that belong to Trusted Domain.
This service call defines a task exception handling routine for the task indicated by tskid according to the content of pk_dtex. If a task exception handling routine has already been defined for the task, this service call updates the definition contents.
When NULL is specified for pk_dtex, the definition of the task exception handling routine for the task is cancelled. At this time, the task pending exception code is cleared to 0, and the task exception handling is disabled.

1) Task ID (tskid)
   Specify the task ID to define a task exception handling routine for tskid. Specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

2) Task exception handling routine attribute (texatr)
   Only TA_HLNG can be specified for texatr.
   - TA_HLNG (= 0x0000)
     Only C-language is supported for task exception handling routine description language.

3) Task exception handling routine start address (texrtn)
   Specify the task exception handling routine start address for texrtn.
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ttextr</code> != <code>TA_HLNG</code></td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>pk_dtex</code> != NULL and <code>ttextr</code> == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid</code> &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tskid</code> &gt; <code>VTMAX_TSK</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>pk_dtex</code> != NULL and the operand-read access to the area indicated by <code>pk_cstk</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by <code>tskid</code> does not exist.</td>
</tr>
</tbody>
</table>
Outline

Raise task exception.

C format

ER ras_tex ( ID tskid, TEXPTN rasptn );
ER iras_tex ( ID tskid, TEXPTN rasptn );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task. TSK_SELF: Invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>TEXPTN rasptn;</td>
<td>Task exception code to be requested.</td>
</tr>
</tbody>
</table>

Explanation

This service call requests task exception handling for the task indicated by tskid. The task pending exception code for the task is ORed with the value indicated by rasptn.

When the conditions for starting task exception handling routine are satisfied, the task exception handling routine is invoked. Please refer to “6.2.3 The starting conditions of task exception handling routines” for the conditions for starting task exception handling routine.

When a task exception handling routine is invoked, the task pending exception code is cleared to 0, and the task exception handling is disabled. The pending exception code before clear and extended information for the task are passed to the task exception handling routine.

At the return from a task exception handling routine, the task exception is enabled, and the task restarts execution from the point immediately before the start of the task exception handling routine.

When a task exception handling routine returns, the RI600PX saves context registers for the task to the user stack. If the user stack is overflow, system goes down.

It is necessary to release from CPU locked state by the end of a task exception handling routine when shifting to the CPU locked state in a task exception handling routine. If CPU is locked at the end of a task exception handling routine, system goes down.

The interrupt priority level (PSW.IPL) before and after the start of a task exception handling routine is not changed. And the interrupt priority level before and after the return from a task exception handling routine is not changed. When the interrupt priority level at the end of a task exception handling routine is higher than the kernel interrupt mask level, system goes down.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
## Macro Value Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.                                                                            <em>rasptn == 0</em></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.                                                                         <em>tskid &lt; 0</em> <em>tskid &gt; VTMAX_TSK</em> <em>When iras_tex was issued from a non-task, TSK_SELF was specified for tskid.</em></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.                                                                               <em>This service call was issued in the CPU locked state.</em> <em>The iras_tex was issued from task.</em> <em>The ras_tex was issued from non-task.</em> *This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ras_tex)                                                    <em>Stack pointer points out of user stack for invoking task.</em></td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.                                                                                <em>Specified task is neither in the WAITING state nor WAITING-SUSPENDED state.</em></td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.                                                                             <em>The task specified by tskid does not exist.</em></td>
</tr>
</tbody>
</table>
### dis_tex

**Outline**
Disable task exception.

**C format**

```c
ER      dis_tex (void);
```

**Parameter(s)**
None.

**Explanation**
This service call disables task exception handling for the invoking task.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A task exception handling routine is not defined for the invoking task.</td>
</tr>
</tbody>
</table>
### ena_tex

#### Outline
Enable task exception.

#### C format
```c
ER ena_tex (void);
```

#### Parameter(s)
None.

#### Explanation
This service call enables task exception handling for the invoking task.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A task exception handling routine is not defined for the invoking task.</td>
</tr>
</tbody>
</table>
sns_tex

Outline
Reference task exception disabled state.

C format

```c
BOOL sns_tex (void);
```

Parameter(s)
None.

Explanation
This service call returns TRUE if the task in the RUNNING state is in the task exception disabled state, and otherwise returns FALSE. Table 19-7 shows the details.

<table>
<thead>
<tr>
<th>Task in the RUNNING state</th>
<th>Task exception handling routine for the task in the RUNNING state</th>
<th>Task exception disabled state for the task in the RUNNING state</th>
<th>Return value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exist</td>
<td>Defined</td>
<td>Enabled state</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disabled state</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not defined</td>
<td>Disabled state</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>When task exception handling routine is not defined, task exception handling is disabled.</td>
</tr>
<tr>
<td>Not exist</td>
<td>-</td>
<td>-</td>
<td>TRUE</td>
<td></td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
<td>Normal completion (task exception disabled state).</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion (task exception enabled state).</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error. - This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
</tbody>
</table>
Outline
Reference task exception state.

C format
ER      ref_tex (ID tskid, T_RTEX *pk_rtex);
ER      iref_tex (ID tskid, T_RTEX *pk_rtex);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td>O</td>
<td>T_RTEX *pk_rtex;</td>
<td>Pointer to the packet returning the task exception state.</td>
</tr>
</tbody>
</table>

[Task exception state packet: T_RTEX]

typedef struct t_rtex {
    STAT    texstat;        /*Task exception handling state*/
    UINT    pndptn;         /*Pending exception code*/
} T_RTEX;

Explanation
Stores task exception state of the task specified by parameter tskid in the area specified by parameter pk_rtex.

- **tskstat**
  Stores task exception handling state.
  
  - **TTEX_ENA**: Task exception enabled state
  - **TTEX_DIS**: Task exception disabled state

- **pndptn**
  Stores the pending exception code.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rtex == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tskid &gt; VTMAX_TSK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When this service call was issued from a non-task, TSK_SELF was specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for tskid.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_tex is issued from task or the ref_tex is issued from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-task, the context error is not detected and normal operation of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_tex)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rtex has not been</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The task specified by tskid does not exist.</td>
</tr>
</tbody>
</table>
19.2.4 Synchronization and communication functions (semaphores)

The following shows the service calls provided by the RI600PX as the synchronization and communication functions (semaphores).

Table 19-8 Synchronization and Communication Functions (Semaphores)

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_sem</td>
<td>Create semaphore</td>
<td>Task</td>
</tr>
<tr>
<td>acre_sem</td>
<td>Create semaphore (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_sem</td>
<td>Delete semaphore</td>
<td>Task</td>
</tr>
<tr>
<td>wai_sem</td>
<td>Acquire semaphore resource (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>pol_sem</td>
<td>Acquire semaphore resource (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>ipol_sem</td>
<td>Acquire semaphore resource (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>twai_sem</td>
<td>Acquire semaphore resource (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>sig_sem</td>
<td>Release semaphore resource</td>
<td>Task</td>
</tr>
<tr>
<td>isig_sem</td>
<td>Release semaphore resource</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_sem</td>
<td>Reference semaphore state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_sem</td>
<td>Reference semaphore state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline
Create semaphore.

C format
ER  cre_sem (ID semid, T_CSEM *pk_csem );
ER_ID acre_sem ( T_CSEM *pk_csem );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID semid;</td>
<td>ID number of the semaphore.</td>
</tr>
<tr>
<td>I</td>
<td>T_CSEM *pk_csem;</td>
<td>Pointer to the packet containing the semaphore creation information.</td>
</tr>
</tbody>
</table>

[Semaphore creation information packet : T_CSEM]

```c
typedef struct t_csem {
    ATR    sematr;         /*Semaphore attribute*/
    UINT   isemcnt;        /*Initial semaphore count*/
    UINT   maxsem;         /*Maximum semaphore count*/
} T_CSEM;
```

Explanation
This service call can be called from tasks that belong to Trusted Domain. The cre_sem creates a semaphore with semaphore ID indicated by semid according to the content of pk_csem. The acre_sem creates a semaphore according to the content of pk_csem, and returns the created semaphore ID.

1 ) Semaphore attribute (sematr)
The following are specified for sematr:
   sematr := ( TA_TFIFO | TA_TPRI )
   - TA_TFIFO ( = 0x0000)
      Task wait queue is managed in FIFO order.
   - TA_TPRI ( = 0x0001)
      Task wait queue is managed in task current priority order. Among tasks with the same priority, they are queued in FIFO order.

2 ) Initial semaphore count (isemcnt)
Specify initial semaphore count within the range from 0 to maxsem.

3 ) Maximum semaphore count (maxsem)
Specify maximum semaphore count within the range from 1 to TMAX_MAXSEM ( = 65535).
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of <code>acre_sem</code>. (Created semaphore ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of <code>cre_sem</code>.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either of bits in <code>sematr</code> except bit0 is 1.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>pk_csem</code> == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>maxsem</code> == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>maxsem</code> &gt; <code>TMAX_MAXSEM</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>maxsem</code> &lt; <code>isemcnt</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for <code>cre_sem</code>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid</code> &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid</code> &gt; <code>VTMAX_SEM</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by <code>pk_csem</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for <code>acre_sem</code>)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for <code>cre_sem</code>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The semaphore specified by <code>semid</code> exists.</td>
</tr>
</tbody>
</table>
### del_sem

**Outline**
Delete semaphore.

**C format**

```
ER      del_sem (ID semid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>ID number of the semaphore.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call can be called from tasks that belong to Trusted Domain.
This service call deletes the semaphore indicated by `semid`.
When there are waiting tasks for the target semaphore by using `wai_sem` or `twai_sem`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `wai_sem` or `twai_sem`.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td>- semid &lt;= 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- semid &gt; VTMAX_SEM</td>
<td></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- This service call was issued from non-task.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;</td>
<td></td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
<td></td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
<td></td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td>- The semaphore specified by <code>semid</code> does not exist.</td>
<td></td>
</tr>
</tbody>
</table>
**Outline**

Acquire semaphore resource (waiting forever).

**C format**

```c
ER  wai_sem (ID semid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>semid;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID number of the semaphore.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call acquires a resource from the semaphore specified by parameter `semid` (subtracts 1 from the semaphore counter).

If no resources are acquired from the target semaphore when this service call is issued (no available resources exist), this service call does not acquire resources but queues the invoking task to the target semaphore wait queue and moves it from the `RUNNING` state to the `WAITING` state (resource acquisition wait state).

The `WAITING` state for a semaphore resource is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Semaphore Resource Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>sig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The resource was released to the target semaphore as a result of issuing <code>isig_sem</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_sem</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

**Note**

Invoking tasks are queued to the target semaphore wait queue in the order specified at creating the semaphore (FIFO order or current priority order).

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid ≤ 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid &gt; VTMAX_SEM</code></td>
</tr>
</tbody>
</table>
### Macro Values

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_CTX   | -25   | Context error.  
|         |       | - This service call was issued from a non-task.  
|         |       | - This service call was issued in the CPU locked state.  
|         |       | - This service call was issued in the dispatching disabled state.  
|         |       | - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.  
| E_MACV  | -26   | Memory access violation.  
|         |       | - Stack pointer points out of user stack for invoking task.  
| E_NOEXS | -42   | Non-existent object.  
|         |       | - The semaphore specified by semid does not exist.  
| E_RLWAI | -49   | Forced release from the WAITING state.  
|         |       | - Accept rel_wai/irel_wai while waiting.  
| E_DLT   | -51   | Waiting object deleted.  
|         |       | - Accept del_sem while waiting.  

pol_sem
ipol_sem

Outline
Acquire semaphore resource (polling).

C format
ER      pol_sem (ID semid);
ER      isem_sem (ID semid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>semid;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID number of the semaphore.</td>
</tr>
</tbody>
</table>

Explanation
This service call acquires a resource from the semaphore specified by parameter semid (subtracts 1 from the semaphore counter). If a resource could not be acquired from the target semaphore (semaphore counter is set to 0) when this service call is issued, the counter manipulation processing is not performed but “E_TMOUT” is returned.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- semid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- semid &gt; VTMAX_SEM</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: When the ipol_sem is issued from task or the pol_sem is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The semaphore specified by semid does not exist.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure.</td>
</tr>
</tbody>
</table>
twai_sem

Outline
Acquire semaphore resource (with time-out).

C format
ER twai_sem (ID semid, TMO tmout);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>semid;</td>
</tr>
<tr>
<td>I</td>
<td>TMO</td>
<td>tmout;</td>
</tr>
</tbody>
</table>

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>semid;</td>
</tr>
<tr>
<td>I</td>
<td>TMO</td>
<td>tmout;</td>
</tr>
</tbody>
</table>

Explanation
This service call acquires a resource from the semaphore specified by parameter semid (subtracts 1 from the semaphore counter).
If no resources are acquired from the target semaphore when service call is issued this (no available resources exist), this service call does not acquire resources but queues the invoking task to the target semaphore wait queue and moves it from the RUNNING state to the WAITING state with time-out (resource acquisition wait state).
The WAITING state for a semaphore resource is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Semaphore Resource Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The resource was released to the target semaphore as a result of issuing sig_sem.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The resource was released to the target semaphore as a result of issuing isig_sem.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_sem while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

Note 1 Invoking tasks are queued to the target semaphore wait queue in the order specified at creating the semaphore (FIFO order or current priority order).

Note 2 TMO_FEVR is specified for wait time tmout, processing equivalent to wai_sem will be executed. When TMO_POL is specified, processing equivalent to pol_sem will be executed.
### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_PAR  | -17   | Parameter error.  
- `tmout < -1`
- `tmout > (0x7FFFFFFF - TIC_NUME) / TIC_DENO`
| E_ID   | -18   | Invalid ID number. 
- `semid ≤ 0`
- `semid > VTMAX_SEM`
| E_CTX  | -25   | Context error.  
- This service call was issued from a non-task.
- This service call was issued in the CPU locked state.
- This service call was issued in the dispatching disabled state.
- This service call was issued in the status "PSW.IPL > kernel interrupt mask level".  
| E_MACV | -26   | Memory access violation.  
- Stack pointer points out of user stack for invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The semaphore specified by `semid` does not exist. |
| E_RLWAI| -49   | Forced release from the WAITING state.  
- Accept `rel_wai/irel_wai` while waiting. |
| E_TMOUT| -50   | Polling failure or specified time has elapsed. |
| E_DLT  | -51   | Waiting object deleted.  
- Accept `del_sem` while waiting. |
**sig_sem**

**isig_sem**

Outline

Release semaphore resource.

C format

```
ER      sig_sem (ID semid);
ER      isig_sem (ID semid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID semid;</td>
<td>ID number of the semaphore.</td>
</tr>
</tbody>
</table>

Explanation

These service calls releases the resource to the semaphore specified by parameter `semid` (adds 1 to the semaphore counter).

If a task is queued in the wait queue of the target semaphore when this service call is issued, the counter manipulation processing is not performed but the resource is passed to the relevant task (first task of wait queue).

As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (WAITING state for a semaphore resource) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

Note  With the RI600PX, the maximum possible number of semaphore resources is defined at semaphore creation.

If the number of resources exceeds the maximum resource count, this service call therefore does not release the acquired resources (addition to the semaphore counter value) but returns E_QOVR.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>semid</code> &gt; VMAX_SEM</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The isig_sem was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The sig_sem was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for sig_sem)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existant object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The semaphore specified by semid does not exist.</td>
</tr>
<tr>
<td>E_QOVR</td>
<td>-43</td>
<td>Queuing overflow.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Resource count exceeded the maximum resource count.</td>
</tr>
</tbody>
</table>
ref_sem
iref_sem

Outline
Reference semaphore state.

C format
ER ref_sem (ID semid, T_RSEM *pk_rsem);
ER iref_sem (ID semid, T_RSEM *pk_rsem);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID semid;</td>
<td>ID number of the semaphore.</td>
</tr>
<tr>
<td>O</td>
<td>T_RSEM *pk_rsem</td>
<td>Pointer to the packet returning the semaphore state.</td>
</tr>
</tbody>
</table>

Semaphore state packet: T_RSEM

typedef struct t_rsem {
    ID      wtskid;         /*Existence of waiting task*/
    UINT    semcnt;         /*Current resource count*/
} T_RSEM;

Explanation
Stores semaphore state packet (ID number of the task at the head of the wait queue, current resource count, etc.) of the semaphore specified by parameter semid in the area specified by parameter pk_rsem.

- **wtskid**
  Stores whether a task is queued to the semaphore wait queue.

  - TSK_NONE: No applicable task
  - Value: ID number of the task at the head of the wait queue

- **semcnt**
  Stores the current resource count.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
</tbody>
</table>
  - pk_rsem == NULL
### Macro Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_ID   | -18   | Invalid ID number.  
|        |       | - semid ≤ 0  
|        |       | - semid > VMAX_SEM |
| E_CTX  | -25   | Context error.  
|        |       | - This service call was issued in the CPU locked state.  
|        |       | - This service call was issued in the status "PSW.IPL > kernel interrupt mask level". |
|        |       | **Note** When the iref_sem is issued from task or the ref_sem is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed. |
| E_MACV | -26   | Memory access violation. (only for ref_sem)  
|        |       | - The operand-write access to the area indicated by pk_rsem has not been permitted to the invoking task. |
| E_NOEXS| -42   | Non-existent object.  
|        |       | - The semaphore specified by semid does not exist. |
19.2.5 Synchronization and communication functions (eventflags)

The following shows the service calls provided by the RI600PX as the synchronization and communication functions (eventflags).

Table 19-9  Synchronization and Communication Functions (Eventflags)

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_flg</td>
<td>Create eventflag</td>
<td>Task</td>
</tr>
<tr>
<td>acre_flg</td>
<td>Create eventflag (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_flg</td>
<td>Delete eventflag</td>
<td>Task</td>
</tr>
<tr>
<td>set_flg</td>
<td>Set eventflag</td>
<td>Task</td>
</tr>
<tr>
<td>iset_flg</td>
<td>Set eventflag</td>
<td>Non-task</td>
</tr>
<tr>
<td>clr_flg</td>
<td>Clear eventflag</td>
<td>Task</td>
</tr>
<tr>
<td>iclr_flg</td>
<td>Clear eventflag</td>
<td>Non-task</td>
</tr>
<tr>
<td>wai_flg</td>
<td>Wait for eventflag (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>pol_flg</td>
<td>Wait for eventflag (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>ipol_flg</td>
<td>Wait for eventflag (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>twai_flg</td>
<td>Wait for eventflag (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>ref_flg</td>
<td>Reference eventflag state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_flg</td>
<td>Reference eventflag state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline

Create eventflag.

C format

```c
ER      cre_flg (ID flgid, T_CFLG *pk_cflg);
ER_ID   acre_flg ( T_CFLG *pk_cflg);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>T_CFLG *pk_cflg;</td>
<td>Pointer to the packet containing the eventflag creation information.</td>
</tr>
</tbody>
</table>

[Eventflag creation information packet : T_CFLG]

```c
typedef struct  t_cflg {
    ATR     flgatr;         /*Eventflag attribute*/
    FLGPTN  iflgptn;        /*Initial bit pattern*/
} T_CFLG;
```

Explanation

This service call can be called from tasks that belong to Trusted Domain. The `cre_flg` creates a eventflag with eventflag ID indicated by `flgid` according to the content of `pk_cflg`. The `acre_flg` creates a eventflag according to the content of `pk_cflg`, and returns the created eventflag ID.

1. Eventflag attribute (flgatr)
The following are specified for `flgatr`.

   - `TA_TFIFO` ( = 0x0000)
     Task wait queue is managed in FIFO order.

   - `TA_TPRI` ( = 0x0001)
     Task wait queue is managed in task current priority order. Among tasks with the same priority, they are queued in FIFO order. When TA_CLR attribute is not specified, even if there is the TA_TPRI attribute specified, the queue is managed in the same way as for the TA_TFIFO attribute. This behavior falls outside µITRON4.0 specification.

   - `TA_WSGL` ( = 0x0000)
     Does not permit multiple tasks to wait for the eventflag.

   - `TA_WMUL` ( = 0x0002)
     Permit multiple tasks to wait for the eventflag.
- TA_CLR ( = 0x0004)
  All the bits of the eventflag are cleared when wai_flg, pol_flg, ipol_flg or twai_flg ends normally.

2 . Initial bit pattern (iflgptn)
  Specify initial eventflag bit pattern.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive value</td>
<td>Normal completion of acre_flg. (Created eventflag ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_flg.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute&lt;br&gt; - Either of bits in flgatr except bit0, bit1 and bit2 is 1.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.&lt;br&gt; - pk_cflg == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_flg)&lt;br&gt; - flgid &lt; 0&lt;br&gt; - flgid &gt; VTMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.&lt;br&gt; - This service call was issued in the CPU locked state.&lt;br&gt; - This service call was issued from non-task.&lt;br&gt; - This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.&lt;br&gt; - Stack pointer points out of user stack for invoking task.&lt;br&gt; - The operand-read access to the area indicated by pk_cflg has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.&lt;br&gt; - The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available.(only for acre_flg)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_flg)&lt;br&gt; - The eventflag specified by flgid exists.</td>
</tr>
</tbody>
</table>
del_flg

Outline
Delete eventflag.

C format

```c
ER del_flg (ID flgid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>flgid;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID number of the eventflag.</td>
</tr>
</tbody>
</table>

Explanation

This service call can be called from tasks that belong to Trusted Domain.
This service call deletes the eventflag indicated by `flgid`.
When there are waiting tasks for the target eventflag by using `wai_flg` or `twai_flg`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `wai_flg` or `twai_flg`.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid &gt; VTMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by <code>flgid</code> does not exist.</td>
</tr>
</tbody>
</table>
set_flg
iset_flg

Outline
Set eventflag.

C format

ER      set_flg (ID flgid, FLGPTN setptn);
ER      iset_flg (ID flgid, FLGPTN setptn);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>FLGPTN setptn;</td>
<td>Bit pattern to set.</td>
</tr>
</tbody>
</table>

Explanation

These service calls set the result of ORing the bit pattern of the eventflag specified by parameter flgid and the bit pattern specified by parameter setptn as the bit pattern of the target eventflag. After that, these service calls evaluate whether the wait condition of the tasks in the wait queue is satisfied. This evaluation is done in order of the wait queue. If the wait condition is satisfied, the relevant task is unlinked from the wait queue at the same time as bit pattern setting processing. As a result, the relevant task is moved from the WAITING state (WAITING state for an eventflag) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. At this time, the bit pattern of the target event flag is cleared to 0 and this service call finishes processing if the TA_CLR attribute is specified for the target eventflag.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid &gt; VMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The iset_flg was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The set_flg was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for set_flg)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by flgid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Clear eventflag.

C format

ER    clr_flg (ID flgid, FLGPTN clrptn);
ER    iclr_flg (ID flgid, FLGPTN clrptn);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>FLGPTN clrptn;</td>
<td>Bit pattern to clear.</td>
</tr>
</tbody>
</table>

Explanation
This service call sets the result of ANDing the bit pattern set to the eventflag specified by parameter flgid and the bit pattern specified by parameter clrptn as the bit pattern of the target eventflag.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flgid &gt; VTMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iclr_flg is issued from task or the clr_flg is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by flgid does not exist.</td>
</tr>
</tbody>
</table>
**wai_flg**

**Outline**
Wait for eventflag (waiting forever).

**C format**
```
ER      wai_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>FLGPTN waiptn;</td>
<td>Wait bit pattern.</td>
</tr>
<tr>
<td>I</td>
<td>MODE wfmode;</td>
<td>Wait mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TWF_ANDW</strong>: AND waiting condition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TWF_ORW</strong>: OR waiting condition.</td>
</tr>
<tr>
<td>O</td>
<td>FLGPTN *p_flgptn;</td>
<td>Bit pattern causing a task to be released from waiting.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call checks whether the bit pattern specified by parameter waiptn and the bit pattern that satisfies the required condition specified by parameter wfmode are set to the eventflag specified by parameter flgid.

If a bit pattern that satisfies the required condition has been set for the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter p_flgptn.

If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue.

As a result, the invoking task is unlinked from the ready queue and is moved from the RUNNING state to the WAITING state (WAITING state for an eventflag).

The WAITING state for an eventflag is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for an Eventflag Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing set_flg.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing iset_flg.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_flg while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

The following shows the specification format of required condition wfmode.

- **wfmode == TWF_ANDW**
  Checks whether all of the bits to which 1 is set by parameter waiptn are set as the target eventflag.
Checks which bit, among bits to which 1 is set by parameter \textit{waiptn}, is set as the target eventflag.

\textbf{Note 1} With the RI600PX, whether to enable queuing of multiple tasks to the event flag wait queue is defined at eventflag creation. If this service call is issued for the event flag (TA\_WSGL attribute) to which a wait task is queued, therefore, “E\_ILUSE” is returned regardless of whether the required condition is immediately satisfied.

\textbf{TA\_WSGL:} Only one task is allowed to be in the WAITING state for the eventflag.

\textbf{TA\_WMUL:} Multiple tasks are allowed to be in the WAITING state for the eventflag.

\textbf{Note 2} Invoking tasks are queued to the target event flag (TA\_WMUL attribute) wait queue in the order specified at creating the eventflag (FIFO order or current priority order). However, when the TA\_CLR attribute is not specified, the wait queue is managed in the FIFO order even if the priority order is specified. This behavior falls outside μITRON4.0 specification.

\textbf{Note 3} The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA\_CLR attribute) is satisfied.

\section*{Return value}

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{waiptn} == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{wfmode} is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{p_flgptn} == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{flgid} &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{flgid} &gt; VTMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW_IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by \textit{p_flgptn} has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- There is already a task waiting for an eventflag with the TA_WSGL attribute.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by \textit{flgid} does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept \textit{rel_wai} while waiting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept \textit{irel_wai} while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept \textit{del_flg} while waiting.</td>
</tr>
</tbody>
</table>
Outline
Wait for eventflag (polling).

C format

```c
ER      pol_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER      ipol_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>FLGPTN waiptn;</td>
<td>Wait bit pattern.</td>
</tr>
</tbody>
</table>
| I   | MODE wfmode;    | Wait mode.  
  * TWF_ANDW: AND waiting condition.  
  * TWF_ORW: OR waiting condition.   |
| O   | FLGPTN *p_flgptn; | Bit pattern causing a task to be released from waiting.                     |

Explanation
This service call checks whether the bit pattern specified by parameter `waiptn` and the bit pattern that satisfies the required condition specified by parameter `wfmode` are set to the eventflag specified by parameter `flgid`.

- If the bit pattern that satisfies the required condition has been set to the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter `p_flgptn`.
- If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, “E_TMOUT” is returned.

The following shows the specification format of required condition `wfmode`.

- `wfmode == TWF_ANDW`  
  Checks whether all of the bits to which 1 is set by parameter `waiptn` are set as the target eventflag.

- `wfmode == TWF_ORW`  
  Checks which bit, among bits to which 1 is set by parameter `waiptn`, is set as the target eventflag.

Note 1 With the RI600PX, whether to enable queuing of multiple tasks to the event flag wait queue is defined during configuration. If this service call is issued for the event flag (TA_WSGL attribute) to which a wait task is queued, therefore, “E_ILUSE” is returned regardless of whether the required condition is immediately satisfied.

- **TA_WSGL:** Only one task is allowed to be in the WAITING state for the eventflag.
- **TA_VMUL:** Multiple tasks are allowed to be in the WAITING state for the eventflag.

Note 2 The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA_CLR attribute) is satisfied.
### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>waitpnt == 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>wfmode</code> is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>p_flgptn == NULL</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>flgid ≤ 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>flgid &gt; VTMAX_FLG</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> When the ipol_flg is issued from task or the pol_flg is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for pol_flg)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>p_flgptn</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- There is already a task waiting for an eventflag with the <code>TA_WSGL</code> attribute.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by <code>flgid</code> does not exist.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure</td>
</tr>
</tbody>
</table>
twai_flg

Outline
Wait for eventflag (with time-out).

C format
ER  twai_flg (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>I</td>
<td>FLGPTN waiptn;</td>
<td>Wait bit pattern.</td>
</tr>
<tr>
<td>I</td>
<td>MODE wfmode;</td>
<td>Wait mode.</td>
</tr>
<tr>
<td></td>
<td>TWF_ANDW: AND waiting condition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TWF_ORW: OR waiting condition.</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>FLGPTN *p_flgptn;</td>
<td>Bit pattern causing a task to be released from waiting.</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
<tr>
<td></td>
<td>TMO_FEVR: Waiting forever.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TMO_POL: Polling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: Specified time-out.</td>
<td></td>
</tr>
</tbody>
</table>

Explanation
This service call checks whether the bit pattern specified by parameter waiptn and the bit pattern that satisfies the required condition specified by parameter wfmode are set to the eventflag specified by parameter flgid.
If a bit pattern that satisfies the required condition has been set for the target eventflag, the bit pattern of the target eventflag is stored in the area specified by parameter p_flgptn.
If the bit pattern of the target eventflag does not satisfy the required condition when this service call is issued, the invoking task is queued to the target eventflag wait queue.
As a result, the invoking task is unlinked from the ready queue and is moved from the RUNNING state to the WAITING state (WAITING state for an eventflag).
The WAITING state for an eventflag is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for an Eventflag Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing set_flag.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A bit pattern that satisfies the required condition was set to the target eventflag as a result of issuing iset_flag.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
</tbody>
</table>
The following shows the specification format of required condition \textit{wfmode}.

- \textit{wfmode} == TWF\_ANDW
  Checks whether all of the bits to which 1 is set by parameter \textit{waiptn} are set as the target eventflag.

- \textit{wfmode} == TWF\_ORW
  Checks which bit, among bits to which 1 is set by parameter \textit{waiptn}, is set as the target eventflag.

Note 1 With the RI600PX, whether to enable queuing of multiple tasks to the event flag wait queue is defined at eventflag creation. If this service call is issued for the event flag (TA\_WSGL attribute) to which a wait task is queued, therefore, “E\_ILUSE” is returned regardless of whether the required condition is immediately satisfied.

\begin{itemize}
  \item [TA\_WSGL:] Only one task is allowed to be in the WAITING state for the eventflag.
  \item [TA\_WMUL:] Multiple tasks are allowed to be in the WAITING state for the eventflag.
\end{itemize}

Note 2 Invoking tasks are queued to the target event flag (TA\_WMUL attribute) wait queue in the order specified at creating the eventflag (FIFO order or current priority order). However, when the TA\_CLR attribute is not specified, the wait queue is managed in the FIFO order even if the priority order is specified. This behavior falls outside μITRON4.0 specification.

Note 3 The RI600PX performs bit pattern clear processing (0 setting) when the required condition of the target eventflag (TA\_CLR attribute) is satisfied.

Note 4 TMO\_FEVR is specified for wait time \textit{tmout}, processing equivalent to \textit{wai_flg} will be executed. When TMO\_POL is specified, processing equivalent to \textit{pol_flg} will be executed.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{waiptn} == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{wfmode} is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{p_flgptn} == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{tmout} &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{tmout} &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{flgid} &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \textit{flgid} &gt; VTMAX_FLG</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW_IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_MACV | -26   | Memory access violation.  
- Stack pointer points out of user stack for invoking task.  
- The operand-write access to the area indicated by `p_flgptn` has not been permitted to the invoking task. |
| E_ILUSE | -28 | Illegal use of service call.  
- There is already a task waiting for an eventflag with the `TA_WSGL` attribute. |
| E_NOEXS | -42 | Non-existent object.  
- The eventflag specified by `flgid` does not exist. |
| E_RLWAI | -49 | Forced release from the WAITING state.  
- Accept `rel_wai`/`irel_wai` while waiting. |
| E_TMOUT | -50 | Polling failure or specified time has elapsed. |
| E_DLT  | -51  | Waiting object deleted.  
- Accept `del_flag` while waiting. |
Outline
Reference eventflag state.

C format

```c
ER ref_flg (ID flgid, T_RFLG *pk_rflg);
ER iref_flg (ID flgid, T_RFLG *pk_rflg);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID flgid;</td>
<td>ID number of the eventflag.</td>
</tr>
<tr>
<td>O</td>
<td>T_RFLG *pk_rflg;</td>
<td>Pointer to the packet returning the eventflag state.</td>
</tr>
</tbody>
</table>

[Eventflag state packet: T_RFLG]

```c
typedef struct t_rflg {
    ID wtskid;         /*Existence of waiting task*/
    FLGPTN flgptn;     /*Current bit pattern*/
} T_RFLG;
```

Explanation
Stores eventflag state packet (ID number of the task at the head of the wait queue, current bit pattern, etc.) of the eventflag specified by parameter flgid in the area specified by parameter pk_rflg.

- **wtskid**
  Stores whether a task is queued to the event flag wait queue.
  - **TSK_NONE**: No applicable task
  - **Value**: ID number of the task at the head of the wait queue

- **flgptn**
  Stores the current bit pattern.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rflg == NULL</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( fglid \leq 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( fglid &gt; \text{VTMAX_FLG} )</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> When the iref_flg is issued from task or the ref_flg is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_flg)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by ( pk_rflg ) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The eventflag specified by ( fglid ) does not exist.</td>
</tr>
</tbody>
</table>
19.2.6 Synchronization and communication functions (data queues)

The following shows the service calls provided by the RI600PX as the synchronization and communication functions (data queues).

Table 19-10  Synchronization and Communication Functions (Data Queues)

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_dtq</td>
<td>Create data queue</td>
<td>Task</td>
</tr>
<tr>
<td>acre_dtq</td>
<td>Create data queue (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_dtq</td>
<td>Delete data queue</td>
<td>Task</td>
</tr>
<tr>
<td>snd_dtq</td>
<td>Send to data queue (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>psnd_dtq</td>
<td>Send to data queue (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>ipsnd_dtq</td>
<td>Send to data queue (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>tsnd_dtq</td>
<td>Send to data queue (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>fsnd_dtq</td>
<td>Forced send to data queue</td>
<td>Task</td>
</tr>
<tr>
<td>ifsnd_dtq</td>
<td>Forced send to data queue</td>
<td>Non-task</td>
</tr>
<tr>
<td>rcv_dtq</td>
<td>Receive from data queue (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>prcv_dtq</td>
<td>Receive from data queue (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>iprcv_dtq</td>
<td>Receive from data queue (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>trcv_dtq</td>
<td>Receive from data queue (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>ref_dtq</td>
<td>Reference data queue state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_dtq</td>
<td>Reference data queue state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline
Create data queue.

C format

```
ER cre_dtq (ID dtqid, T_CDTQ *pk_cdtq);
ER_ID acre_dtq ( T_CDTQ *pk_cdtq);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>T_CDTQ *pk_cdtq;</td>
<td>Pointer to the packet containing the data queue creation information.</td>
</tr>
</tbody>
</table>

[Data queue creation information packet : T_CDTQ]

```
typedef struct t_cdtq {
    ATR    dtqatr; /*Data queue attribute*/
    UINT   dtqcnt; /*Capacity of the data queue area (the number of data elements)*/
    VP     dtq;    /*Start address of the data queue area*/
} T_CDTQ;
```

Explanation
This service call can be called from tasks that belong to Trusted Domain. The cre_dtq creates a data queue with data queue ID indicated by dtqid according to the content of pk_cdtq. The acre_dtq creates a data queue according to the content of pk_cdtq, and returns the created data queue ID.

1) Data queue attribute (dtqatr)
The following are specified for dtqatr.
- `TA_TFIFO` (= 0x0000)
  Task wait queue for sending is managed in FIFO order.
- `TA_TPRI` (= 0x0001)
  Task wait queue for sending is managed in task current priority order. Among tasks with the same priority, they are queued in FIFO order.

   Note Task wait queue for receiving is managed in FIFO order.

2) Capacity of the data queue area (dtqcnt), Start address of the data queue area (dtq)
The application acquires TSZ_DTQ(dtqcnt) bytes of data queue area and specifies the start address for dtq. It is also possible to specify 0 as dtqcnt. In this case, since data cannot be stored in the data queue, the data sending task or data receiving task that has performed its operation first will enter the WAITING state. The WAITING
state of that task is canceled when the task of another side has performed its operation. Thus, data sending tasks and data receiving tasks are completely synchronized. Note, dtq is disregarded when dtqcnt is 0.

Note 1 For details of TSZ_DTQ macro, refer to “18.3.2 Macros for Data Queue”.

Note 2 The RI600PX is not concerned of anything of the access permission to the data queue area. Usually, the data queue area should be generated to the area other than memory objects and user stacks. When the data queue area is generated in the memory object, a task with the operand-write access permission to the memory object might rewrite data queue area by mistake.

Note 3 The μITRON 4.0 specification defines the function that the kernel allocates data queue area when NULL is specified for dtq. But RI600PX does not support this function.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive value</td>
<td>Normal completion of acre_dtq. (Created data queue ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_dtq.</td>
</tr>
</tbody>
</table>
| E_RSATR | -11   | Reserved attribute  
|        |       | - Either of bits in dtqatr except bit0 is 1. |
| E_PAR | -17   | Parameter error.  
|        |       | - pk_cdtq == NULL  
|        |       | - dtqcnt > 65535  
|        |       | - dtqcnt != 0 and dtq + TSZ_DTQ(dtqcnt) > 0x100000000 |
| E_ID  | -18   | Invalid ID number. (only for cre_dtq)  
|        |       | - dtqid < 0  
|        |       | - dtqid > VTMAX_DTQ |
| E_CTX | -25   | Context error.  
|        |       | - This service call was issued in the CPU locked state.  
|        |       | - This service call was issued from non-task.  
|        |       | - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV | -26    | Memory access violation.  
|        |       | - Stack pointer points out of user stack for invoking task.  
|        |       | - The operand-read access to the area indicated by pk_cdtq has not been permitted to the invoking task. |
| E_OACV | -27    | Object access violation.  
|        |       | - The invoking task does not belong to trusted domain. |
| E_NOMEM | -33   | Insufficient memory.  
|        |       | - dtqcnt != 0 and dtq == NULL |
| E_NOID | -34    | No ID number available. (only for acre_dtq) |
| E_OBJ | -41    | Object state error. (only for cre_dtq)  
|        |       | - The data queue specified by dtqid exists. |
**del_dtq**

**Outline**
Delete data queue.

**C format**

```c
ER del_dtq (ID dtqid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call can be called from tasks that belong to Trusted Domain. This service call deletes the data queue indicated by `dtqid`.

When there are waiting tasks for the target data queue by using `snd_dtq`, `tsnd_dtq`, `rcv_dtq` or `trcv_dtq`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `snd_dtq`, `tsnd_dtq`, `rcv_dtq` or `trcv_dtq`.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid</code> &gt; VTMAX_DTQ</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by <code>dtqid</code> does not exist.</td>
</tr>
</tbody>
</table>
Outline

Send to data queue (waiting forever).

C format

```c
ER  snd_dtq (ID dtqid, VP_INT data);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>VP_INT data;</td>
<td>Data element to be sent to the data queue.</td>
</tr>
</tbody>
</table>

Explanation

This service call processes as follows according to the situation of the data queue specified by the parameter `dtqid`.

- There is a task in the reception wait queue.
  
  This service call transfers the data specified by parameter `data` to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.
  
  This service call stores the data specified by parameter `data` to the data queue.

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.
  
  This service call queues the invoking task to the transmission wait queue of the target data queue and moves it from the RUNNING state to the WAITING state (data transmission wait state).

  The sending WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>rcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>prcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>iprcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing <code>trcv_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The data queue is reset as a result of issuing <code>vrst_dtq</code>.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_dtq</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

Note 1  Data is written to the data queue area in the order of the data transmission request.
Note 2  Invoking tasks are queued to the transmission wait queue of the target data queue in the order specified at creating the data queue (FIFO order or current priority order).

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid &gt; VTMAX_DTQ</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by dtqid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_dtq while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_dtq)</td>
</tr>
</tbody>
</table>
Outline
Send to data queue (polling).

C format
ER  psnd_dtq (ID dtqid, VP_INT data);
ER  ipsnd_dtq (ID dtqid, VP_INT data);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>VP_INT data;</td>
<td>Data element to be sent to the data queue.</td>
</tr>
</tbody>
</table>

Explanation
These service calls process as follows according to the situation of the data queue specified by the parameter dtqid.
- There is a task in the reception wait queue.
  These service calls transfer the data specified by parameter data to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.
  These service calls store the data specified by parameter data to the data queue.
- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.
  These service calls return “E_TMOUT”.

Note Data is written to the data queue area of the target data queue in the order of the data transmission request.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid &gt; VTMAX_DTQ</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- The ipsnd_dtq was issued from task.  
- The psnd_dtq was issued from non-task.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV | -26   | Memory access violation. (only for psnd_dtq)  
- Stack pointer points out of user stack for invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The data queue specified by dtqid does not exist. |
| E_TMOUT| -50   | Polling failure. |
Outline
Send to data queue (with time-out).

C format
ER tsnd_dtq (ID dtqid, VP_INT data, TMO tmout);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>VP_INT data;</td>
<td>Data element to be sent to the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TMO_FEVR:</strong> Waiting forever.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TMO_POL:</strong> Polling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value:</strong> Specified time-out.</td>
</tr>
</tbody>
</table>

Explanation
This service call processes as follows according to the situation of the data queue specified by the parameter dtqid.

- There is a task in the reception wait queue.  
  This service call transfers the data specified by parameter data to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the data queue.  
  This service call stores the data specified by parameter data to the data queue.

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the data queue, or there is a task in the transmission wait queue.  
  This service call queues the invoking task to the transmission wait queue of the target data queue and moves it from the RUNNING state to the WAITING state with time (data transmission wait state).

The sending WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the data queue area as a result of issuing rcv_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing prcv_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing iprcv_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the data queue area as a result of issuing trcv_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The data queue is reset as a result of issuing vrst_dtq.</td>
<td>EV_RST</td>
</tr>
</tbody>
</table>
Note 1  Data is written to the data queue area of the target data queue in the order of the data transmission request.

Note 2  Invoking tasks are queued to the transmission wait queue of the target data queue in the order specified at creating the data queue (FIFO order or current priority order).

Note 3  TMO_FEV R is specified for wait time tmout, processing equivalent to snd_dtq will be executed. When TMO_POL is specified, processing equivalent to psnd_dtq will be executed.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &gt; ((0x7FFFFFFF - TIC_NUME) / TIC_DENO)</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid \leq 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid \geq VTMAX_DTQ</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by dtqid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_dtq while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_dtq)</td>
</tr>
</tbody>
</table>
Outline
Forced send to data queue.

C format
ER fsnd_dtq (ID dtqid, VP_INT data);
ER ifsnd_dtq (ID dtqid, VP_INT data);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>VP_INT data;</td>
<td>Data element to be sent to the data queue.</td>
</tr>
</tbody>
</table>

Explanation
These service calls process as follows according to the situation of the data queue specified by the parameter dtqid.

- There is a task in the reception wait queue.
This service call transfers the data specified by parameter data to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (data reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue.
This service call stores the data specified by parameter data to the data queue. If there is no available space in the data queue, this service call deletes the oldest data in the data queue before storing the data specified by data to the data queue.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid &gt; VMAX_DTQ</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- The ifsnd_dtq was issued from task.  
- The fsnd_dtq was issued from non-task.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV | -26   | Memory access violation. (only for fsnd_dtq)  
- Stack pointer points out of user stack for invoking task. |
| E_ILUSE| -28   | Illegal use of service call.  
- The capacity of the data queue area is 0. |
| E_NOEXS| -42   | Non-existent object.  
- The data queue specified by dtqid does not exist. |
**rcv_dtq**

Outline
Receive from data queue (waiting forever).

C format

```c
ER rcv_dtq (ID dtqid, VP_INT *p_data);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>O</td>
<td>VP_INT *p_data;</td>
<td>Data element received from the data queue.</td>
</tr>
</tbody>
</table>

**Explanation**
This service call processes as follows according to the situation of the data queue specified by the parameter `dtqid`.

- There is a data in the data queue.
  This service call takes out the oldest data from the data queue and stores the data to the area specified by `p_data`. When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.

- There is no data in the data queue and there is a task in the transmission wait queue.
  This service call stores the data specified by the task in the top of the transmission wait queue to the area specified by `p_data`. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. Note, this situation is caused only when the capacity of the data queue is 0.

- There is no data in the data queue and there is no task in the transmission wait queue.
  This service call queues the invoking task to the reception wait queue of the target data queue and moves it from the RUNNING state to the WAITING state (data reception wait state). The receiving WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data was sent to the data queue area as a result of issuing <code>snd_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <code>psnd_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <code>ipsnd_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <code>tsnd_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing <code>fsnd_dtq</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_dtq</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>
Note Invoking tasks are queued to the reception wait queue of the target data queue in the order of the data reception request.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( p_data == NULL )</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( dtqid \leq 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( dtqid &gt; VTMAX_DTQ )</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by ( p_data ) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by ( dtqid ) does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept ( rel_wai/irel_wai ) while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept ( del_dtq ) while waiting.</td>
</tr>
</tbody>
</table>
Receive from data queue (polling).

C format

```c
ER  prcv_dtq (ID dtqid, VP_INT *p_data);
ER  iprcv_dtq (ID dtqid, VP_INT *p_data);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dttid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>O</td>
<td>VP_INT *p_data;</td>
<td>Data element received from the data queue.</td>
</tr>
</tbody>
</table>

Explanation

These service calls process as follows according to the situation of the data queue specified by the parameter dtqid.

- There is a data in the data queue.
  This service call takes out the oldest data from the data queue and stores the data to the area specified by p_data. When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.

- There is no data in the data queue and there is a task in the transmission wait queue.
  These service calls store the data specified by the task in the top of the transmission wait queue to the area specified by p_data. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.
  Note, this situation is caused only when the capacity of the data queue is 0.

- There is no data in the data queue and there is no task in the transmission wait queue.
  These service calls return "E_TMOUT".

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_data == NULL</td>
</tr>
</tbody>
</table>
### Macro Value Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_ID    | -18   | Invalid ID number.  
|         |       | - $dtqid \leq 0$  
|         |       | - $dtqid > \text{VTMAX DTQ}$  |
| E_CTX   | -25   | Context error.  
|         |       | - This service call was issued in the CPU locked state.  
|         |       | - The iprcv_dtq was issued from task.  
|         |       | - The prcv_dtq was issued from non-task.  
|         |       | - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.  |
| E_MACV  | -26   | Memory access violation. (only for prcv_dtq)  
|         |       | - Stack pointer points out of user stack for invoking task.  
|         |       | - The operand-write access to the area indicated by $p_{\text{data}}$ has not been permitted to the invoking task.  |
| E_NOEXS | -42   | Non-existent object.  
|         |       | - The data queue specified by $dtqid$ does not exist.  |
| E_TMOUT | -50   | Polling failure.  |
trcv_dtq

Outline
Receive from data queue (with time-out).

C format

```
ER  trcv_dtq (ID dtqid, VP_INT *p_data, TMO tmout);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>O</td>
<td>VP_INT *p_data;</td>
<td>Data element received from the data queue.</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
</tbody>
</table>

Explanation
This service call processes as follows according to the situation of the data queue specified by the parameter dtqid.

- There is a data in the data queue.
  This service call takes out the oldest data from the data queue and stores the data to the area specified by p_data. When there is a task in the transmission wait queue, this service call stores the data sent by the task in the top of the transmission wait queue and moves it from the WAITING state (data transmission wait state) to the READY state.

- There is no data in the data queue and there is a task in the transmission wait queue.
  This service call stores the data specified by the task in the top of the transmission wait queue to the area specified by p_data. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (data transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. Note, this situation is caused only when the capacity of the data queue is 0.

- There is no data in the data queue and there is no task in the transmission wait queue.
  This service call queues the invoking task to the reception wait queue of the target data queue and moves it from the RUNNING state to the WAITING state with time (data reception wait state). The receiving WAITING state for a data queue is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Data Queue Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data was sent to the data queue area as a result of issuing snd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing psnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing ipsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing tsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing fsnd_dtq.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Data was sent to the data queue area as a result of issuing ifsnd_dtq.</td>
<td>E_OK</td>
</tr>
</tbody>
</table>
Note 1 Invoking tasks are queued to the reception wait queue of the target data queue in the order of the data reception request.

Note 2 TMO_FEVR is specified for wait time tmout, processing equivalent to rcv_dtq will be executed. When TMO_POL is specified, processing equivalent to prcv_dtq will be executed.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_data == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- dtqid &gt; VTMAX_DTQ</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level“.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_data has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by dtqid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_dtq while waiting.</td>
</tr>
</tbody>
</table>
Outline
Reference data queue state.

C format

```c
ER ref_dtq (ID dtqid, T_RDTQ *pk_rdtq);
ER iref_dtq (ID dtqid, T_RDTQ *pk_rdtq);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID dtqid;</td>
<td>ID number of the data queue.</td>
</tr>
<tr>
<td>O</td>
<td>T_RDTQ *pk_rdtq;</td>
<td>Pointer to the packet returning the data queue state.</td>
</tr>
</tbody>
</table>

[Data queue state packet: T_RDTQ]

```c
typedef struct t_rdtq {
    ID stskid;         /*Existence of tasks waiting for data transmission*/
    ID rtskid;         /*Existence of tasks waiting for data reception*/
    UINT sdtqcnt;        /*Number of data elements in data queue*/
} T_RDTQ;
```

Explanation
These service calls store the detailed information of the data queue (existence of waiting tasks, number of data elements in the data queue, etc.) specified by parameter dtqid into the area specified by parameter pk_rdtq.

- **stskid**
  Stores whether a task is queued to the transmission wait queue of the data queue.
  
  **TSK_NONE:** No applicable task
  Value: ID number of the task at the head of the transmission wait queue

- **rtskid**
  Stores whether a task is queued to the reception wait queue of the data queue.
  
  **TSK_NONE:** No applicable task
  Value: ID number of the task at the head of the reception wait queue

- **sdtqcnt**
  Stores the number of data elements in data queue.
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>pk_rdtq == NULL</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid &lt; 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid &gt; VMAX_DTQ</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>Note</td>
<td></td>
<td>When the <code>iref_dtq</code> is issued from task or the <code>ref_dtq</code> is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for <code>ref_dtq</code>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>pk_rdtq</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by <code>dtqid</code> does not exist.</td>
</tr>
</tbody>
</table>
19.2.7 Synchronization and communication functions (mailboxes)

The following shows the service calls provided by the RI600PX as the synchronization and communication functions (mailboxes).

Table 19-11 Synchronization and Communication Functions (Mailboxes)

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_mbx</td>
<td>Create mailbox</td>
<td>Task</td>
</tr>
<tr>
<td>acre_mbx</td>
<td>Create mailbox (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_mbx</td>
<td>Delete mailbox</td>
<td>Task</td>
</tr>
<tr>
<td>snd_mbx</td>
<td>Send to mailbox</td>
<td>Task</td>
</tr>
<tr>
<td>isnd_mbx</td>
<td>Send to mailbox</td>
<td>Non-task</td>
</tr>
<tr>
<td>rcv_mbx</td>
<td>Receive from mailbox (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>prcv_mbx</td>
<td>Receive from mailbox (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>iprcv_mbx</td>
<td>Receive from mailbox (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>trcv_mbx</td>
<td>Receive from mailbox (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>ref_mbx</td>
<td>Reference mailbox state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_mbx</td>
<td>Reference mailbox state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
cre_mbx
acre_mbx

Outline
Create mailbox.

C format
ER      cre_mbx (ID mbxid, T_CMBX *pk_cmbx );
ER_ID   acre_mbx ( T_CMBX *pk_cmbx );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>I</td>
<td>T_CMBX *pk_cmbx;</td>
<td>Pointer to the packet containing the mailbox creation information.</td>
</tr>
</tbody>
</table>

[Mailbox creation information packet : T_CMBX]

typedef struct t_cmbx {
    ATR   mbxatr;  /*Mailbox attribute*/
    PRI   maxmpri; /*Maximum message priority*/
    VP    mprihd;  /*For future expansion*/
} T_CMBX;

Explanation
This service call can be called from tasks that belong to Trusted Domain.
The cre_mbx creates a mailbox with mailbox ID indicated by mbxid according to the content of pk_cmbx. The acre_mbx creates a mailbox according to the content of pk_cmbx, and returns the created mailbox ID.

1) Mailbox attribute (mbxatr)
The following are specified for mbxatr.

mbxatr := ( ( TA_TFIFO || TA_TPRI ) | ( TA_MFIFO || TA_MPRI ) )
- TA_TFIFO (= 0x0000)
  Task wait queue is managed in FIFO order.
- TA_TPRI (= 0x0001)
  Task wait queue is managed in task current priority order. Among tasks with the same priority, they are queued in FIFO order.
- TA_MFIFO (= 0x0000)
  Message queue is managed in FIFO order.
- TA_MPRI (= 0x0002)
  Message queue is managed in message priority order. Among messages with the same priority, they are queued in FIFO order.
2) Maximum message priority (maxmpri)
   When TA_MPRI is specified for mbxatr, the range of message priority which can be used is from 1 to maxmpri. Ranges of the value that can be specified are from 1 to TMAX_MPRI.

3) mprihd
   The mprihd is for future expansion, and is only disregarded.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_mbx. (Created mailbox ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_mbx.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either of bits in mbxatr except bit0 and bit1 is 1.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_cmbx == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When TA_MPRI is specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- maxmpri &lt;= 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- maxmpri &gt; TMAX_MPRI</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_mbx)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &gt; VTMAX_MBX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pk_cmbx has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_mbx)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_mbx)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by mbxid exists.</td>
</tr>
</tbody>
</table>
del_mbx

Outline
Delete mailbox.

C format
ER del_mbx (ID mbxid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mbxid;</td>
</tr>
<tr>
<td></td>
<td>mbxid</td>
<td>ID number of the mailbox.</td>
</tr>
</tbody>
</table>

Explanation
This service call can be called from tasks that belong to Trusted Domain. This service call deletes the mailbox indicated by mbxid. When there are waiting tasks for the target mailbox by using rcv_mbx or trcv_mbx, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the rcv_mbx or trcv_mbx.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &gt; VTMAX_MBX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by mbxid does not exist.</td>
</tr>
</tbody>
</table>
**snd_mbx**

**isnd_mbx**

### Outline

Send to mailbox.

### C format

```c
ER      snd_mbx (ID mbxid, T_MSG *pk_msg);
ER      isnd_mbx (ID mbxid, T_MSG *pk_msg);
```

### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>I</td>
<td>T_MSG *pk_msg;</td>
<td>Start address of the message packet to be sent to the mailbox.</td>
</tr>
</tbody>
</table>

[Message packet T_MSG for TA_MFIFO attribute]

```c
typedef struct {
    VP      msghead;       /*RI600PX management area*/
} T_MSG;
```

[Message packet for T_MSG_PRI for TA_MPRI attribute]

```c
typedef struct {
    T_MSG   msgque;        /*Message header*/
    PRI     msgpri;        /*Message priority*/
} T_MSG_PRI;
```

### Explanation

This service call transmits the message specified by parameter `pk_msg` to the mailbox specified by parameter `mbxid` (queues the message in the wait queue).

If a task is queued to the target mailbox wait queue when this service call is issued, the message is not queued but handed over to the relevant task (first task of the wait queue).

As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (receiving WAITING state for a mailbox) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

**Note 1** Messages are queued to the target mailbox message queue in the order specified at creating the mailbox (FIFO order or message priority order).

**Note 2** Do not modify transmitted message (the area indicated by `pk_msg`) until the message is received.
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{pk_msg} == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- When the target mailbox has TA_MPRI attribute:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{msgpri} \leq 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{msgpri} &gt; TMAX_MPRI</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{mbxid} \leq 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- \texttt{mbxid} &gt; VTMAX_MBX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The isnd_mbx was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The snd_mbx was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for snd_mbx)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read and operand-write access to the message header area has not been permitted to the invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message header area:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TA_MFIFO attribute : The T_MSG structure started from the address indicated by \texttt{pk_msg}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TA_MPRI attribute : The T_MSG_PRI structure started from the address indicated by \texttt{pk_msg}</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by \texttt{mbxid} does not exist.</td>
</tr>
</tbody>
</table>
# rcv_mbx

## Outline
Receive from mailbox (waiting forever).

## C format
```c
ER rcv_mbx (ID mbxid, T_MSG **ppk_msg);
```

## Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>O</td>
<td>T_MSG **ppk_msg;</td>
<td>Start address of the message packet received from the mailbox.</td>
</tr>
</tbody>
</table>

### [Message packet T_MSG for TA_MFIFO attribute ]

```c
typedef struct {
    VP msghead;    /*RI600PX management area*/
} T_MSG;
```

### [Message packet T_MSG_PRI for TA_MPRI attribute]

```c
typedef struct {
    T_MSG msgque;    /*Message header*/
    PRI msgpri;      /*Message priority*/
} T_MSG_PRI;
```

## Explanation
This service call receives a message from the mailbox specified by parameter `mbxid`, and stores its start address in the area specified by parameter `ppk_msg`. If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service call does not receive messages but queues the invoking task to the target mailbox wait queue and moves it from the RUNNING state to the WAITING state (message reception wait state). The receiving WAITING state for a mailbox is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Mailbox Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>snd_mbx</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>isnd_mbx</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mbx</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>
Note: Invoking tasks are queued to the target mailbox wait queue in the order specified at creating the mailbox (FIFO order or current priority order).

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ppk_msg</code> == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbxid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbxid</code> &gt; <code>VTMAX_MBX</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>ppk_msg</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by <code>mbxid</code> does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept <code>rel_wai</code>/<code>irel_wai</code> while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept <code>del_mbx</code> while waiting.</td>
</tr>
</tbody>
</table>
Outline
Receive from mailbox (polling).

C format

```
ER      prcv_mbx (ID mbxid, T_MSG **ppk_msg);
ER      iprcv_mbx (ID mbxid, T_MSG **ppk_msg);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>O</td>
<td>T_MSG **ppk_msg;</td>
<td>Start address of the message packet received from the mailbox.</td>
</tr>
</tbody>
</table>

[M][Message packet T_MSG for TA_MFIFO attribute ]

```
typedef struct {
    VP      msghead;       /*RI600PX management area*/
} T_MSG;
```

[Message packet T_MSG_PRI for TA_MPRI attribute]

```
typedef struct {
    T_MSG   msgque;        /*Message header*/
    PRI     msgpri;        /*Message priority*/
} T_MSG_PRI;
```

Explanation
This service call receives a message from the mailbox specified by parameter mbxid, and stores its start address in the area specified by parameter ppk_msg.

If the message could not be received from the target mailbox (no messages were queued in the wait queue) when this service call is issued, message reception processing is not executed but “E_TMOUT” is returned.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_PAR  | -17   | parameter error.  
|        |       | - ppk_msg == NULL |
| E_ID   | -18   | Invalid ID number.  
|        |       | - mbxid < 0  
|        |       | - mbxid > VTMAX_MBX |
| E_CTX  | -25   | Context error.  
|        |       | - This service call was issued in the CPU locked state.  
|        |       | - This service call was issued in the status "PSW.IPL > kernel interrupt mask level".  
|        |       | Note When the iprcv_mbx is issued from task or the prcv_mbx is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed. |
| E_MACV | -26   | Memory access violation. (only for prcv_mbx)  
|        |       | - The operand-write access to the area indicated by ppk_msg has not been permitted to the invoking task. |
| E_NOEXS| -42   | Non-existent object.  
|        |       | - The mailbox specified by mbxid does not exist. |
| E_TMOUT| -50   | Polling failure. |
**trcv_mbx**

### Outline
Receive from mailbox (with time-out).

### C format
```
ER     trcv_mbx (ID mbxid, T_MSG **ppk_msg, TMO tmout);
```

### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>O</td>
<td>T_MSG **ppk_msg;</td>
<td>Start address of the message packet received from the mailbox.</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
</tbody>
</table>

**[Message packet: T_MSG]**

```c
typedef struct t_msg {
    struct t_msg *msgnext; /*Reserved for future use*/
} T_MSG;
```

**[Message packet: T_MSG_PRI]**

```c
typedef struct t_msg_pri {
    struct t_msg msgque; /*Reserved for future use*/
    PRI msgpri; /*Message priority*/
} T_MSG_PRI;
```

### Explanation
This service call receives a message from the mailbox specified by parameter `mbxid`, and stores its start address in the area specified by parameter `ppk_msg`.

If no message could be received from the target mailbox (no messages were queued to the wait queue) when this service call is issued, this service call does not receive messages but queues the invoking task to the target mailbox wait queue and moves it from the RUNNING state to the WAITING state with time-out (message reception wait state). The receiving WAITING state for a mailbox is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Receiving WAITING State for a Mailbox Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>snd_mbx</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A message was transmitted to the target mailbox as a result of issuing <code>isnd_mbx</code>.</td>
<td>E_OK</td>
</tr>
</tbody>
</table>
Note 1 Invoking tasks are queued to the target mailbox wait queue in the order specified at creating the mailbox (FIFO order or current priority order).

Note 2 TMO_FEVR is specified for wait time \( tmout \), processing equivalent to \( rcv_mbx \) will be executed. When TMO_POL is specified, processing equivalent to \( prcv_mbx \) will be executed.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( ppk_msg == ) NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( tmout &lt; -1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DEN)</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( mbxid &lt; 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ( mbxid &gt; VTMAX_MBX )</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by ( ppk_msg ) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by ( mbxid ) does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mbx while waiting.</td>
</tr>
</tbody>
</table>
**ref_mbx**  
**iref_mbx**

### Outline
Reference mailbox state.

### C format
```c
ER ref_mbx (ID mbxid, T_RMBX *pk_rmbx);
ER iref_mbx (ID mbxid, T_RMBX *pk_rmbx);
```

### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbxid;</td>
<td>ID number of the mailbox.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMBX *pk_rmbx;</td>
<td>Pointer to the packet returning the mailbox state.</td>
</tr>
</tbody>
</table>

**[Mailbox state packet: T_RMBX]**

```c
typedef struct t_rmbx {
  ID      wtskid;         /*Existence of waiting task*/
  T_MSG   *pk_msg;        /*Existence of waiting message*/
} T_RMBX;
```

### Explanation
Stores mailbox state packet (ID number of the task at the head of the wait queue, start address of the message packet at the head of the wait queue) of the mailbox specified by parameter `mbxid` in the area specified by parameter `pk_rmbx`.

- **wtskid**  
  Stores whether a task is queued to the mailbox wait queue.
  - **TSK_NONE**: No applicable task  
    Value: ID number of the task at the head of the wait queue

- **pk_msg**  
  Stores whether a message is queued to the mailbox wait queue.
  - **NULL**: No applicable message  
    Value: Start address of the message packet at the head of the wait queue

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rmbx == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &gt; VTMAX_MBX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_mbx is issued from task or the ref_mbx is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_mbx)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rmbx has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by mbxid does not exist.</td>
</tr>
</tbody>
</table>
19.2.8 Extended synchronization and communication functions (mutexes)

The following shows the service calls provided by the RI600PX as the extended synchronization and communication functions (mutexes).

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_mtx</td>
<td>Create mutex</td>
<td>Task</td>
</tr>
<tr>
<td>acre_mtx</td>
<td>Create mutex (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_mtx</td>
<td>Delete mutex</td>
<td>Task</td>
</tr>
<tr>
<td>loc_mtx</td>
<td>Lock mutex (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>ploc_mtx</td>
<td>Lock mutex (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>tloc_mtx</td>
<td>Lock mutex (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>unl_mtx</td>
<td>Unlock mutex</td>
<td>Task</td>
</tr>
<tr>
<td>ref_mtx</td>
<td>Reference mutex state</td>
<td>Task</td>
</tr>
</tbody>
</table>
Outline
Create mutex.

C format

```c
ER      cre_mtx (ID mtxid, T_CMTX *pk_c mtx);
ER_ID   acre_mtx ( T_CMTX *pk_c mtx);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>ID number of the mutex.</td>
</tr>
<tr>
<td>I</td>
<td>T_CMTX</td>
<td>Pointer to the packet containing the mutex creation information.</td>
</tr>
</tbody>
</table>

[Mutex creation information packet : T_CMTX]

```c
typedef struct t_c mtx {
    ATR    mtxatr;        /* Mutex attribute */
    PRI    ceilpri;       /* Ceiling priority */
} T_CMTX;
```

Explanation

This service call can be called from tasks that belong to Trusted Domain. The `cre_mtx` creates a mutex with mutex ID indicated by `mtxid` according to the content of `pk_c mtx`. The `acre_mtx` creates a mutex according to the content of `pk_c mtx`, and returns the created mutex ID.

1) Mutex attribute (`mtxatr`)
   - Only `TA_CEILING` can be specified for `mtxatr`.
   - `TA_CEILING` (= 0x0003)
     - Priority ceiling protocol
     - For details, refer to "8.2.3 Simplified priority ceiling protocol".

   Note Task wait queue is managed in task current priority order. Note, tasks of the same current priority are managed in FIFO order.

2) Ceiling priority (`ceilpri`)
   - The current task priority of the task which locks a mutex rises to the `ceilpri`.
   - Ranges of the value that can be specified are from 1 to `TMAX_TPRI`. 
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_mtx. (Created mutex ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_mtx.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute  &lt;br&gt; - mtxatr != TA_CEILING.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.  &lt;br&gt; - pk_cmtx == NULL  &lt;br&gt; - ceilpri &lt;= 0  &lt;br&gt; - ceilpri &gt; TMAX_TPRI</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_mtx)  &lt;br&gt; - mtxid &lt; 0  &lt;br&gt; - mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.  &lt;br&gt; - This service call was issued in the CPU locked state.  &lt;br&gt; - This service call was issued from non-task.  &lt;br&gt; - This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.  &lt;br&gt; - Stack pointer points out of user stack for invoking task.  &lt;br&gt; - The operand-read access to the area indicated by pk_cmtx has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.  &lt;br&gt; - The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_mtx)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_mtx)  &lt;br&gt; - The mutex specified by mtxid exists.</td>
</tr>
</tbody>
</table>
### del_mtx

#### Outline
Delete mutex.

#### C format
```
ER   del_mtx (ID mtxid);
```

#### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mtxid; ID number of the mutex.</td>
</tr>
</tbody>
</table>

#### Explanation
This service call can be called from tasks that belong to Trusted Domain. This service call deletes the mutex indicated by `mtxid`.

When either of task locks the target mutex, the lock by the task is cancelled. As a result, the current task priority of the task is returned to the base priority when there is no mutex being locked by the task. The task is not notified that the mutex has been deleted. If an attempt is later made to unlock the mutex by using ` unl_mtx`, an error E_NOEXS is returned.

When there are waiting tasks for the target mutex by using `loc_mtx` or `tloc_mtx`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `loc_mtx` or `tloc_mtx`.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
</tbody>
</table>
### Macro Value Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by <code>mtxid</code> does not exist.</td>
</tr>
</tbody>
</table>
loc_mtx

Outline
Lock mutex (waiting forever).

C format
ER loc_mtx (ID mtxid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mtxid;</td>
<td>ID number of the mutex.</td>
</tr>
</tbody>
</table>

Explanation
This service call locks the mutex specified by parameter mtxid.
If the target mutex could not be locked (another task has been locked) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to the WAITING state (mutex wait state).
The WAITING state for a mutex is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Mutex Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing uni_mtx.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing ext_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing exd_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing ter_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_mtx while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the invoking task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

Note 1 Invoking tasks are queued to the target mutex wait queue in task current priority order. Among tasks with the same priority, they are queued in FIFO order.
Note 2 This service call returns “E_ILUSE” if this service call is re-issued for the mutex that has been locked by the invoking task (multiple-locking of mutex).
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task has already locked the target mutex.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ceiling priority violation (the base priority of the invoking task &lt; the ceiling priority of the target mutex)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by mtxid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mtx while waiting.</td>
</tr>
</tbody>
</table>
**ploc_mtx**

**Outline**
Lock mutex (polling).

**C format**

```c
ER      ploc_mtx (ID mtxid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mtxid;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID</td>
<td>ID number of the mutex.</td>
</tr>
</tbody>
</table>

**Explanation**
This service call locks the mutex specified by parameter mtxid.
If the target mutex could not be locked (another task has been locked) when this service call is issued but “E_TMOOUT” is returned.
When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the invoking task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

Note: This service call returns “E_ILUSE” if this service call is re-issued for the mutex that has been locked by the invoking task (multiple-locking of mutex).

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task has already locked the target mutex.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ceiling priority violation (the base priority of the invoking task &lt; the ceiling priority of the target mutex)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by \textit{mtxid} does not exist.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure.</td>
</tr>
</tbody>
</table>
tloc_mtx

Outline
Lock mutex (with time-out).

C format
ER tloc_mtx (ID mtxid, TMO tmout);  

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mtxid;</td>
</tr>
<tr>
<td>I</td>
<td>TMO</td>
<td>tmout;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_FEVR:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_POL:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value:</td>
</tr>
</tbody>
</table>

Explanation
This service call locks the mutex specified by parameter mtxid. If the target mutex could not be locked (another task has been locked) when this service call is issued, this service call queues the invoking task to the target mutex wait queue and moves it from the RUNNING state to the WAITING state with time-out (mutex wait state). The WAITING state for a mutex is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Mutex Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing unl_mtx.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing ext_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing exd_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The locked state of the target mutex was cancelled as a result of issuing ter_tsk.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_mtx while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

When the mutex is locked, this service call changes the current priority of the invoking task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the invoking task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

Note 1 Invoking tasks are queued to the target mutex wait queue in task current priority order. Among tasks with the same priority, they are queued in FIFO order.
Note 2 This service call returns “E_ILUSE” if this service call is re-issued for the mutex that has been locked by the invoking task (multiple-locking of mutex).

Note 3 TMO_FEVR is specified for wait time tmout, processing equivalent to loc_mtx will be executed. When TMO_POL is specified, processing equivalent to ploc_mtx will be executed.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task has already locked the target mutex.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ceiling priority violation (the base priority of the invoking task &lt; the ceiling priority of the target mutex)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by mtxid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mtx while waiting.</td>
</tr>
</tbody>
</table>
Outline
Unlock mutex.

C format
ER unl_mtx (ID mtxid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mtxid;</td>
</tr>
</tbody>
</table>

Explanation
This service call unlocks the locked mutex specified by parameter mtxid.
If a task has been queued to the target mutex wait queue when this service call is issued, mutex lock processing is performed by the task (the first task in the wait queue) immediately after mutex unlock processing.
As a result, the task is unlinked from the wait queue and moves from the WAITING state (mutex wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. And this service call changes the current priority of the task to the ceiling priority of the target mutex. However, this service call does not change the current priority when the task has locked other mutexes and the ceiling priority of the target mutex is lower than or equal to the ceiling priority of the locked mutexes.

Note 1 A locked mutex can be unlocked only by the task that locked the mutex.
If this service call is issued for a mutex that was not locked by the invoking task, “E_ILUSE” is returned.

Note 2 When a task terminates, mutexes locked by the task are unlocked.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task have not locked the target mutex.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by ( mtid ) does not exist.</td>
</tr>
</tbody>
</table>
**ref_mtx**

**Outline**
Reference mutex state.

**C format**

```c
ER ref_mtx (ID mtxid, T_RMTX *pk_rmtx);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mtxid; ID number of the mutex.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMTX</td>
<td>*pk_rmtx; Pointer to the packet returning the mutex state.</td>
</tr>
</tbody>
</table>

[Mutex state packet: T_RMTX]

```c
typedef struct t_rmtx {
    ID htskid; /*Existence of locked mutex*/
    ID wtskid; /*Existence of waiting task*/
} T_RMTX;
```

**Explanation**
This service call stores the detailed information of the mutex specified by parameter *mtxid* (existence of locked mutexes, waiting tasks, etc.) into the area specified by parameter *pk_rmtx*.

- **htskid**
  Stores whether a task that is locking a mutex exists.
  
  **TSK_NONE:** No applicable task  
  **Value:** ID number of the task locking the mutex

- **wtskid**
  Stores whether a task is queued to the mutex wait queue.
  
  **TSK_NONE:** No applicable task  
  **Value:** ID number of the task at the head of the wait queue

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>parameter error.</td>
</tr>
</tbody>
</table>
  
  - *pk_rmtx == NULL*
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid \leq 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mtxid &gt; VTMAX_MTX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rmtx has not been permitted to the invoking task</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mutex specified by mtxid does not exist.</td>
</tr>
</tbody>
</table>
19.2.9 Extended synchronization and communication functions (message buffers)

The following shows the service calls provided by the RI600PX as the extended synchronization and communication functions (message buffers).

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_mbf</td>
<td>Create message buffer</td>
<td>Task</td>
</tr>
<tr>
<td>acre_mbf</td>
<td>Create message buffer (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_mbf</td>
<td>Delete message buffer</td>
<td>Task</td>
</tr>
<tr>
<td>snd_mbf</td>
<td>Send to message buffer (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>psnd_mbf</td>
<td>Send to message buffer (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>ipsnd_mbf</td>
<td>Send to message buffer (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>tsnd_mbf</td>
<td>Send to message buffer (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>rcv_mbf</td>
<td>Receive from message buffer (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>prcv_mbf</td>
<td>Receive from message buffer (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>trcv_mbf</td>
<td>Receive from message buffer (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>ref_mbf</td>
<td>Reference message buffer state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_mbf</td>
<td>Reference message buffer state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>

Table 19-13 Extended Synchronization and Communication Functions (Message Buffers)
Outline

Create message buffer.

C format

```
ER cre_mbf (ID mbfid, T_CMBF *pk_cmbf);
ER_ID acre_mbf ( T_CMBF *pk_cmbf);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>I</td>
<td>T_CMBF *pk_cmbf;</td>
<td>Pointer to the packet containing the message buffer creation information.</td>
</tr>
</tbody>
</table>

[Message buffer creation information packet: T_CMBF]

```
typedef struct t_cmbf {
    ATR mbfatr;    /*Message buffer attribute*/
    UINT maxmsz;   /*Maximum message size (in bytes)*/
    UINT mbfsz;    /*Size of the message buffer area (in bytes)*/
    VP mbf;        /*Start address of the message buffer area*/
} T_CMBF;
```

Explanation

This service call can be called from tasks that belong to Trusted Domain. The cre_mbf creates a message buffer with message buffer ID indicated by mbfid according to the content of pk_cmbf. The acre_mbf creates a message buffer according to the content of pk_cmbf, and returns the created message buffer ID.

1) Message buffer attribute (mbfatr)

Only TA_TFIFO can be specified for mbfatr.

- TA_TFIFO ( = 0x0000)
  Task wait queue for sending is managed in FIFO order.

  Note Task wait queue for receiving is managed in FIFO order.

2) Maximum message size (maxmsz)

Specify the maximum size of message which can be sent to this message buffer. The size of the reception area specified by rcv_mbf, prcv_mbf and trcv_mbf must be not less than maxmsz.

3) Size of the message buffer area (mbfsz), Start address of the message buffer area (mbf)

The application acquires mbfsz bytes of message buffer area and specifies the start address for mbf. It is also possible to specify 0 as mbfsz. In this case, since message cannot be stored in the message buffer, the
message sending task or message receiving task that has performed its operation first will enter the WAITING state. The WAITING state of that task is canceled when the task of another side has performed its operation. Thus, message sending tasks and message receiving tasks are completely synchronized. Note, mbf is disregarded when mbfsz is 0.

Note 1 The RI600PX is not concerned of anything of the access permission to the message buffer area. Usually, the message buffer area should be generated to the area other than memory objects and user stacks. When the message buffer area is generated in the memory object, a task with the operand-write access permission to the memory object might rewrite message buffer area by mistake.

Note 2 The μITRON4.0 specification defines the function that the kernel allocates message buffer area when NULL is specified for mbf. But RI600PX does not support this function.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_mbf. (Created message buffer ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_mbf.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfatr != TA_TFIFO</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pkcmbf == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- maxmsz == 0, maxmsz &gt; 65528</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 &lt; mbfsz &lt; 8, mbfsz &gt; 65532</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfsz == 0 and mbf + mbfsz &gt; 0x100000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfsz == 0 and mbfsz &lt; maxmsz + VTSZ_MBFTBL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_mbf)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &gt; VTMX_MBF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pkcmbf has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOMEM</td>
<td>-33</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfsz != 0 and mbf == NULL</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_mbf)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_mbf)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid exists.</td>
</tr>
</tbody>
</table>
Summary

Delete message buffer.

C format

```
ER del_mbf (ID mbfid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mbfid; ID number of the message buffer.</td>
</tr>
</tbody>
</table>

Explanation

This service call can be called from tasks that belong to Trusted Domain. This service call deletes the message buffer indicated by `mbfid`. When there are waiting tasks for the target message buffer by using `snd_mbf`, `tsnd_mbf`, `rcv_mbf` or `trcv_mbf`, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the `snd_mbf`, `tsnd_mbf`, `rcv_mbf` or `trcv_mbf`.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &gt; VMAX_MBF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Send to message buffer (waiting forever).

C format

```c
ER tsnd_mbf (ID mbfid, VP msg, UINT msgsz);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>I</td>
<td>VP msg;</td>
<td>Pointer to the message to be sent.</td>
</tr>
<tr>
<td>I</td>
<td>UINT msgsz;</td>
<td>Message size to be sent (in bytes).</td>
</tr>
</tbody>
</table>

Explanation
This service call processes as follows according to the situation of the message buffer specified by the parameter mbfid.

- There is a task in the reception wait queue.
  This service call transfers the message specified by parameter msg to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer.
  This service call stores the message specified by parameter msg to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  \[ \text{The amount of decrease} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL} \]

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue.
  This service call queues the invoking task to the transmission wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state (message transmission wait state).
  The sending WAITING state for a message buffer is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing rcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing prcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing trcv_mbf.</td>
<td>E_OK</td>
</tr>
</tbody>
</table>
Note 1 Message is written to the message buffer area in the order of the message transmission request.
Note 2 Invoking tasks are queued to the transmission wait queue of the target message buffer in the FIFO order.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- msgsz == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- msgsz &gt; (Maximum message size specified at creation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- msg == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &gt; VTIME_MBFS</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by msg (start : msg, size : msgsz) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mbf while waiting.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_mbf)</td>
</tr>
</tbody>
</table>
Outline
Send to message buffer (polling).

C format

| ER | psnd_mbf (ID mbfid, VP msg, UINT msgsz); |
| ER | ipsnd_mbf (ID mbfid, VP msg, UINT msgsz); |

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>I</td>
<td>VP msg;</td>
<td>Pointer to the message to be sent.</td>
</tr>
<tr>
<td>I</td>
<td>UINT msgsz;</td>
<td>Message size to be sent (in bytes).</td>
</tr>
</tbody>
</table>

Explanation

These service calls process as follows according to the situation of the message buffer specified by the parameter mbfid.

- There is a task in the reception wait queue.
  This service call transfers the message specified by parameter msg to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer.
  This service call stores the message specified by parameter msg to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  \[ \text{The amount of decrease} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBFTBL} \]

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue.
  These service calls return “E_TMOOUT”.

Note Message is written to the message buffer area in the order of the message transmission request.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_PAR  | -17   | Parameter error.  
- `msgsz == 0`  
- `msgsz > (Maximum message size specified at creation)`  
- `msg == NULL` |
| E_ID   | -18   | Invalid ID number.  
- `mbfid < 0`  
- `mbfid > VTMAX_MBF` |
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- The ipsnd_mbf was issued from task.  
- The psnd_mbf was issued from non-task.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level” |
| E_MACV | -26   | Memory access violation. (only for psnd_mbf)  
- Stack pointer points out of user stack for invoking task.  
- The operand-read access to the area indicated by `msg` (start : `msg`, size : `msgsz`) has not been permitted to the invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The message buffer specified by `mbfid` does not exist. |
| E_TMOUT| -50   | Polling failure. |
Outline
Send to message buffer (with time-out).

C format
```
ER tsnd_mbf (ID mbfid, VP msg, UINT msgsz, TMO tmout);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>I</td>
<td>VP msg;</td>
<td>Pointer to the message to be sent.</td>
</tr>
<tr>
<td>I</td>
<td>UINT msgsz;</td>
<td>Message size to be sent (in bytes).</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
<tr>
<td></td>
<td>TMO_FEVR:</td>
<td>Waiting forever.</td>
</tr>
<tr>
<td></td>
<td>TMO_POL:</td>
<td>Polling.</td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Specified time-out.</td>
</tr>
</tbody>
</table>

Explanation
This service call processes as follows according to the situation of the message buffer specified by the parameter mbfid.

- There is a task in the reception wait queue.
  This service call transfers the message specified by parameter msg to the task in the top of the reception wait queue. As a result, the task is unlinked from the reception wait queue and moves from the WAITING state (message reception wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

- There is no task neither in the reception wait queue and transmission wait queue and there is available space in the message buffer.
  This service call stores the message specified by parameter msg to the message buffer. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.
  \[\text{The amount of decrease} = \text{up4}(\text{msgsz}) + \text{VTSZ_MBTBL}\]

- There is no task neither in the reception wait queue and transmission wait queue and there is no available space in the message buffer, or there is a task in the transmission wait queue.
  This service call queues the invoking task to the transmission wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state with time (message transmission wait state). The sending WAITING state for a message buffer is cancelled in the following cases.

<table>
<thead>
<tr>
<th>Sending WAITING State for a Message Buffer Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing rcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing prcv_mbf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Available space was secured in the message buffer area as a result of issuing trcv_mbf.</td>
<td>E_OK</td>
</tr>
</tbody>
</table>
Note 1  Message is written to the message buffer area in the order of the message transmission request.

Note 2  Invoking tasks are queued to the transmission wait queue of the target message buffer in the FIFO order.

Note 3  TMO_FEVR is specified for wait time tmout, processing equivalent to snd_mbf will be executed. When TMO_POL is specified, processing equivalent to psnd_mbf will be executed.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_PAR | -17   | Parameter error.  
|       |       |  - msgsz == 0  
|       |       |  - msgsz > (Maximum message size specified at creation)  
|       |       |  - msg == NULL  
|       |       |  - tmout < -1  
|       |       |  - tmout > (0x7FFFFFFF - TIC_NUME) / TIC_DENO  |
| E_ID  | -18   | Invalid ID number.  
|       |       |  - mbfid < 0  
|       |       |  - mbfid > VTMAX_MBF  |
| E_CTX | -25   | Context error.  
|       |       |  - This service call was issued from a non-task.  
|       |       |  - This service call was issued in the CPU locked state.  
|       |       |  - This service call was issued in the dispatching disabled state.  
|       |       |  - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.  |
| E_MACV| -26   | Memory access violation.  
|       |       |  - Stack pointer points out of user stack for invoking task.  
<p>|       |       |  - The operand-read access to the area indicated by msg (start : msg, size : msgsz) has not been permitted to the invoking task.  |</p>
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mbf while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_mbf)</td>
</tr>
</tbody>
</table>
rcv_mbf

Outline
Receive from message buffer (waiting forever).

C format

```c
ER_UINT rcv_mbf (ID mbfid, VP msg);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>O</td>
<td>VP msg;</td>
<td>Pointer to store the message.</td>
</tr>
</tbody>
</table>

Explanation

This service call processes as follows according to the situation of the message buffer specified by the parameter `mbfid`.

- There is a message in the message buffer.
  This service call takes out the oldest message from the message buffer and stores the message to the area specified by `msg` and return the size of the message. As a result, the size of available space in the target message buffer increases by the amount calculated by the following expression.

  The amount of increase = up4( Return value ) + VTSZ_MBFTBL

  In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it becomes impossible to store the message in the message buffer.

- When there is a task in the transmission wait queue and there is available space in the message buffer for the message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  The amount of decrease =up4( The message size sent by the task ) + VTSZ_MBFTBL

- There is no message in the message buffer and there is a task in the transmission wait queue.
  This service call stores the message specified by the task in the top of the transmission wait queue to the area pointed by the parameter `msg`. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

  Note, this situation is caused only when the size of the message buffer is 0.

- There is no message in the message buffer and there is no task in the transmission wait queue.
  This service call queues the invoking task to the reception wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state (message reception wait state).

Receiving WAITING State for a Message Buffer Cancel Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message was sent to the message buffer area as a result of issuing <code>snd_mbf</code>.</td>
<td>E_OK</td>
</tr>
</tbody>
</table>
Note 1  The maximum message size is defined at creating the message buffer. The size of the area pointed by \textit{msg} must be not less than the maximum message size.

Note 2  Invoking tasks are queued to the reception wait queue of the target message buffer in the order of the message reception request.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion (the size of the received message).</td>
</tr>
</tbody>
</table>
| E_PAR | -17  | Parameter error.  
\hspace{1em} - \textit{msg} == NULL |
| E_ID  | -18  | Invalid ID number.  
\hspace{1em} - \textit{mbfid} \leq 0  
\hspace{1em} - \textit{mbfid} > \text{VTMAX_MBF} |
| E_CTX | -25  | Context error.  
\hspace{1em} - This service call was issued from a non-task.  
\hspace{1em} - This service call was issued in the CPU locked state.  
\hspace{1em} - This service call was issued in the dispatching disabled state.  
\hspace{1em} - This service call was issued in the status \text{“PSW.IPL > kernel interrupt mask level”}. |
| E_MACV| -26  | Memory access violation.  
\hspace{1em} - Stack pointer points out of user stack for invoking task.  
\hspace{1em} - The operand-write access to the area indicated by \textit{msg} (\text{start} : \textit{msg}, \text{size} : maximum message size specified at creation) has not been permitted to the invoking task. |
| E_NOEXS| -42 | Non-existent object.  
\hspace{1em} - The message buffer specified by \textit{mbfid} does not exist. |
| E_RLWAI| -49 | Forced release from the WAITING state.  
\hspace{1em} - Accept \textit{rel_wai} while \textit{irel_wai} while waiting. |
| E_DLT | -51  | Waiting object deleted.  
\hspace{1em} - Accept \textit{del_mbf} while waiting. |
prcv_mbf

Outline

Receive from message buffer (polling).

C format

```
ER_UINT prcv_mbf (ID mbfid, VP msg);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>O</td>
<td>VP msg;</td>
<td>Pointer to store the message.</td>
</tr>
</tbody>
</table>

Explanation

- There is a message in the message buffer.
  This service call takes out the oldest message from the message buffer and stores the message to the area specified
  by msg and return the size of the message. As a result, the size of available space in the target message buffer
  increases by the amount calculated by the following expression.
  
  \[ \text{The amount of increase} = \text{up4( Return value )} + \text{VTSZ_MBFTBL} \]

  In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it
  becomes impossible to store the message in the message buffer.
  
  - When there is a task in the transmission wait queue and there is available space in the message buffer for the
    message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmis-
    sion wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or
    from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the
    target message buffer decreases by the amount calculated by the following expression.
    
    \[ \text{The amount of decrease} = \text{up4( The message size sent by the task )} + \text{VTSZ_MBFTBL} \]

- There is no message in the message buffer and there is a task in the transmission wait queue.
  This service call stores the message specified by the task in the top of the transmission wait queue to the area
  pointed by the parameter msg. As a result, the task is unlinked from the transmission wait queue and moves from the
  WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to
  the SUSPENDED state.
  Note, this situation is caused only when the size of the message buffer is 0.

- There is no message in the message buffer and there is no task in the transmission wait queue.
  This service call returns "E_TMOUT".

Note  The maximum message size is defined at creating the message buffer. The size of the area pointed by msg
must be not less than the maximum message size.
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion (the size of the received message).</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>msg == NULL</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbfid &lt; 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbfid &gt; VTMAX_MBF</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>msg</code> (start : <code>msg</code>, size : maximum message size specified at creation) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by <code>mbfid</code> does not exist.</td>
</tr>
<tr>
<td>E_TMOOUT</td>
<td>-50</td>
<td>Polling failure.</td>
</tr>
</tbody>
</table>
trcv_mbf

Outline
Receive from message buffer (with time-out).

C format
ER_UINT trcv_mbf (ID mbfid, VP msg, TMO tmout);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
<tr>
<td>O</td>
<td>VP msg;</td>
<td>Pointer to store the message.</td>
</tr>
<tr>
<td>I</td>
<td>TMO tmout;</td>
<td>Specified time-out (in millisecond).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_FEVR: Waiting forever.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMO_POL: Polling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: Specified time-out.</td>
</tr>
</tbody>
</table>

Explanation
This service call processes as follows according to the situation of the message buffer specified by the parameter mbfid.

- There is a message in the message buffer. This service call takes out the oldest message from the message buffer and stores the message to the area specified by msg and return the size of the message. As a result, the size of available space in the target message buffer increases by the amount calculated by the following expression.

  \[ \text{The amount of increase} = \text{up4( Return value )} + \text{VTSZ_MBFSTBL} \]

  In addition, this service call repeats the following processing until task in the transmission wait queue is lost or it becomes impossible to store the message in the message buffer.

- When there is a task in the transmission wait queue and there is available space in the message buffer for the message specified by the task in the top of the transmission wait queue, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. As a result, the size of available space in the target message buffer decreases by the amount calculated by the following expression.

  \[ \text{The amount of decrease} = \text{up4( The message size sent by the task )} + \text{VTSZ_MBFSTBL} \]

- There is no message in the message buffer and there is a task in the transmission wait queue. This service call stores the message specified by the task in the top of the transmission wait queue to the area pointed by the parameter msg. As a result, the task is unlinked from the transmission wait queue and moves from the WAITING state (message transmission wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state. Note, this situation is caused only when the size of the message buffer is 0.

- There is no message in the message buffer and there is no task in the transmission wait queue. This service call queues the invoking task to the reception wait queue of the target message buffer and moves it from the RUNNING state to the WAITING state with time (message reception wait state). The receiving WAITING state for a message buffer is cancelled in the following cases.
Note 1 The maximum message size is defined at creating the message buffer. The size of the area pointed by `msg` must be not less than the maximum message size.

Note 2 Invoking tasks are queued to the reception wait queue of the target message buffer in the order of the message reception request.

Note 3 `TMO_FEVR` is specified for wait time `tmout`, processing equivalent to `rcv_mbf` will be executed. When `TMO_POL` is specified, processing equivalent to `prcv_mbf` will be executed.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion (the size of the received message).</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>msg</code> == <code>NULL</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tmout</code> &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>tmout</code> &gt; <code>(0x7FFFFFFF - TIC_NUME) / TIC_DENO</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbfid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mbfid</code> &gt; <code>VTMAX_MBF</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>msg</code> (start : <code>msg</code>, size : maximum message size specified at creation) has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by <code>mbfid</code> does not exist.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mbf while waiting.</td>
</tr>
</tbody>
</table>
Outline

Reference message buffer state.

C format

```c
ER      ref_mbf (ID mbfid, T_RMBF *pk_rmbf);
ER      iref_mbf (ID mbfid, T_RMBF *pk_rmbf);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMBF *pk_rmbf;</td>
<td>Pointer to the packet returning the message buffer state.</td>
</tr>
</tbody>
</table>

[Message buffer state packet: T_RMBF]

```c
typedef struct t_rmbf {
    ID      stskid; /*Existence of tasks waiting for message transmission*/
    ID      rtskid; /*Existence of tasks waiting for message reception*/
    UINT    smsgcnt; /*Number of message elements in message buffer*/
    SIZE    fmbfsz; /*Available buffer size*/
} T_RMBF;
```

Explanation

These service calls store the detailed information of the message buffer (existence of waiting tasks, number of data elements in the message buffer, etc.) specified by parameter `mbfid` into the area specified by parameter `pk_rmbf`.

- **stskid**
  Stores whether a task is queued to the transmission wait queue of the message buffer.
  - **TSK_NONE**: No applicable task
    - **Value**: ID number of the task at the head of the transmission wait queue

- **rtskid**
  Stores whether a task is queued to the reception wait queue of the message buffer.
  - **TSK_NONE**: No applicable task
    - **Value**: ID number of the task at the head of the reception wait queue

- **smsgcnt**
  Stores the number of message elements in message buffer.

- **fmbfsz**
  Stores available size of the message buffer (in bytes).
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rmbf == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &gt; VTMAX_MBF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_mbf is issued from task or the ref_mbf is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_mbf)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rmbf has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid does not exist.</td>
</tr>
</tbody>
</table>
19.2.10 Memory pool management functions (fixed-sized memory pools)

The following shows the service calls provided by the Ri600PX as the memory pool management functions (fixed-sized memory pools).

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_mpf</td>
<td>Create fixed-sized memory pool</td>
<td></td>
</tr>
<tr>
<td>acre_mpf</td>
<td>Create fixed-sized memory pool (automatic ID assignment)</td>
<td></td>
</tr>
<tr>
<td>del_mpf</td>
<td>Delete fixed-sized memory pool</td>
<td></td>
</tr>
<tr>
<td>get_mpf</td>
<td>Acquire fixed-sized memory block (waiting forever)</td>
<td></td>
</tr>
<tr>
<td>pget_mpf</td>
<td>Acquire fixed-sized memory block (polling)</td>
<td></td>
</tr>
<tr>
<td>ipget_mpf</td>
<td>Acquire fixed-sized memory block (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>tget_mpf</td>
<td>Acquire fixed-sized memory block (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>rel_mpf</td>
<td>Release fixed-sized memory block</td>
<td>Task</td>
</tr>
<tr>
<td>irel_mpf</td>
<td>Release fixed-sized memory block</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_mpf</td>
<td>Reference fixed-sized memory pool state</td>
<td></td>
</tr>
<tr>
<td>iref_mpf</td>
<td>Reference fixed-sized memory pool state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline
Create fixed-sized memory pool.

C format

ER cre_mpf (ID mpfid, T_CMPF *pk_cmpf);
ER_ID acre_mpf ( T_CMPF *pk_cmpf);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mpfid;</td>
<td>ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>T_CMPF *pk_cmpf;</td>
<td>Pointer to the packet containing the fixed-sized memory pool creation information.</td>
</tr>
</tbody>
</table>

[Fixed-sized memory pool creation information packet : T_CMPF]

typedef struct  t_cmpf {
    ATR    mpfatr;  /*fixed-sized memory pool attribute*/
    UINT   blkcnt;  /*Total number of memory blocks*/
    UINT   blksz;   /*Memory block size (in bytes)*/
    VP     mpf;     /*Start address of the fixed-sized memory pool area*/
    VP     mpfmb;   /*Start address of the fixed-sized memory pool management area*/
} T_CMPF;

Explanation
This service call can be called from tasks that belong to Trusted Domain.
The cre_mpf creates a fixed-sized memory pool with fixed-sized memory pool ID indicated by mpfid according to the content of pk_cmpf. The acre_mpf creates a fixed-sized memory pool according to the content of pk_cmpf, and returns the created fixed-sized memory pool ID.

1) Fixed-sized memory pool attribute (mpfatr)
The following are specified for mpfatr.
   mpfatr := ( TA_TFIFO | TA_TPRI )
   - TA_TFIFO (= 0x0000)
     Task wait queue is managed in FIFO order.
   - TA_TPRI (= 0x0001)
     Task wait queue is managed in task current priority order. Among tasks with the same priority, they are queued in FIFO order.

2) Total number of memory blocks (blkcnt), memory block size (blksz), Start address of the fixed-sized memory pool area (mpf)
The application acquires TSZ_MPF( blkcnt, blksz ) bytes of fixed-sized memory pool area and specifies the start address for mpf.

Note 1 For details of TSZ_MPF macro, refer to "18.3.3 Macros for Fixed-sized Memory Pool".

Note 2 The RI600PX is not concerned of anything of the access permission to the fixed-sized memory pool area. To access to the memory block from task, the memory pool area must be in the memory object with appropriate permission. Note, the RI600PX generates management tables in the memory pool area. If the management table is rewritten by allocation, correct system operation cannot be guaranteed.

Note 3 The \uTRON4.0 specification defines the function that the kernel allocates fixed-sized memory pool area when NULL is specified for mpf. But RI600PX does not support this function.

Note 4 The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at fixed-sized memory pool creation.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

3) Start address of the fixed-sized memory pool management area (mpfmb)

The application acquires TSZ_MPFMB( blkcnt, blksz ) bytes of fixed-sized memory pool management area and specifies the start address for mpfmb.

Note 1 For details of TSZ_MPFMB macro, refer to "18.3.3 Macros for Fixed-sized Memory Pool".

Note 2 The RI600PX is not concerned of anything of the access permission to the fixed-sized memory pool management area. Usually, the fixed-sized memory pool management area should be generated to the area other than memory objects and user stacks. When the fixed-sized memory pool management area is generated in the memory object, a task with the operand-write access permission to the memory object might rewrite the fixed-sized memory pool management area by mistake.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_mpf. (Created fixed-sized memory pool ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_mpf.</td>
</tr>
</tbody>
</table>
| E_RSATR | -11  | Reserved attribute  
- Either of bits in mpfatr except bit0 is 1. |
| E_PAR | -17   | Parameter error.  
- pk_cmpf == NULL  
- blkcnt == 0, blkcnt > 65535  
- blksz == 0, blksz > 65535  
- TSZ_MPF(blkcnt, blksz) > VTMAX_AREASIZE  
- mpf + TSZ_MPF(blkcnt, blksz) > 0x100000000 |
| E_ID  | -18   | Invalid ID number. (only for cre_mpf)  
- mpfid < 0  
- mpfid > VTMAX_MPF |
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_CTX    | -25   | Context error.  
|          |       | - This service call was issued in the CPU locked state.  
|          |       | - This service call was issued from non-task.  
|          |       | - This service call was issued in the status "PSW.IPL > kernel interrupt mask level". |
| E_MACV   | -26   | Memory access violation.  
|          |       | - Stack pointer points out of user stack for invoking task.  
|          |       | - The operand-read access to the area indicated by \texttt{pk\_cmpf} has not been permitted to the invoking task. |
| E_OACV   | -27   | Object access violation.  
|          |       | - The invoking task does not belong to trusted domain. |
| E_NOMEM  | -33   | Insufficient memory.  
|          |       | - \texttt{mpf == NULL}  
|          |       | - \texttt{mpfmb == NULL} |
| E_NOID   | -34   | No ID number available.(only for acre\_mpf) |
| E_OBJ    | -41   | Object state error. (only for cre\_mpf)  
|          |       | - The fixed-sized memory pool specified by \texttt{mpfid} exists. |
Outline
Delete fixed-sized memory pool.

C format

```c
ER del_mpf (ID mpfid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mpfid;</td>
<td>ID number of the fixed-sized memory pool.</td>
</tr>
</tbody>
</table>

Explanation

This service call can be called from tasks that belong to Trusted Domain.
This service call deletes the fixed-sized memory pool indicated by mpfid.
When there are waiting tasks for the target fixed-sized memory pool by using get_mpf or tget_mpf, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the get_mpf or tget_mpf.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid &gt; VTMAX_MPF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by mpfid does not exist.</td>
</tr>
</tbody>
</table>
**get_mpf**

**Outline**
Acquire fixed-sized memory block (waiting forever).

**C format**
```
ER      get_mpf (ID mpfid, VP *p_blk);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mpfid; ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>O</td>
<td>VP</td>
<td>*p_blk; Start address of the acquired memory block.</td>
</tr>
</tbody>
</table>

**Explanation**
This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter `mpfid` and stores the start address in the area specified by parameter `p_blk`. If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (no available fixed-size memory blocks exist) when this service call is issued, this service call does not acquire the fixed-size memory block but queues the invoking task to the target fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state (fixed-size memory block acquisition wait state).

The WAITING state for a fixed-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Fixed-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>rel_mpf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing <code>irel_mpf</code>.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>rel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>irel_wai</code> while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The fixed-sized memory pool is reset as a result of issuing <code>vrst_mpf</code>.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>Forced release from waiting (accept <code>del_mpf</code> while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

**Note 1** Invoking tasks are queued to the target fixed-size memory pool wait queue in the order defined at creating the fixed-sized memory pool. (FIFO order or current priority order).

**Note 2** The contents of the block are undefined.

**Note 3** The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_PAR | -17   | Parameter error.  
|       |       | - `p_blk == NULL` |
| E_ID  | -18   | Invalid ID number.  
|       |       | - `mpfid <= 0`  
|       |       | - `mpfid > VTMAX_MPF` |
| E_CTX | -25   | Context error.  
|       |       | - This service call was issued from a non-task.  
|       |       | - This service call was issued in the CPU locked state.  
|       |       | - This service call was issued in the dispatching disabled state.  
|       |       | - This service call was issued in the status “PSW.IPL > kernel interrupt mask level” |
| E_MACV| -26   | Memory access violation.  
|       |       | - Stack pointer points out of user stack for invoking task.  
|       |       | - The operand-write access to the area indicated by `p_blk` has not been permitted to the invoking task. |
| E_NOEXS| -42  | Non-existent object.  
|       |       | - The fixed-sized memory pool specified by `mpfid` does not exist. |
| E_RLWAI| -49  | Forced release from the WAITING state.  
|       |       | - Accept `rel_wai`/`irel_wai` while waiting. |
| E_DLT | -51   | Waiting object deleted.  
|       |       | - Accept `del_mpf` while waiting. |
| EV_RST| -127  | Released from WAITING state by the object reset (`vrst_mpf`) |
Outline

Acquire fixed-sized memory block (polling).

C format

ER    pget_mpf (ID mpfid, VP *p_blk);
ER    ipget_mpf (ID mpfid, VP *p_blk);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mpfid;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>O</td>
<td>VP</td>
<td>*p_blk;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start address of the acquired memory block.</td>
</tr>
</tbody>
</table>

Explanation

This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter mpfid and stores the start address in the area specified by parameter p_blk. If a fixed-sized memory block could not be acquired from the target fixed-sized memory pool (no available fixed-sized memory blocks exist) when this service call is issued, fixed-sized memory block acquisition processing is not performed but "E_TMOUT" is returned.

Note 1 The contents of the block are undefined.

Note 2 The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_blk == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid &gt; VTMAX_MPF</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the ipget_mpf is issued from task or the pget_mpf is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for pget_mpf)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_blk has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by mpfid does not exist.</td>
</tr>
<tr>
<td>E_TMOOUT</td>
<td>-50</td>
<td>Polling failure.</td>
</tr>
</tbody>
</table>
### tget_mpf

**Outline**

Acquire fixed-sized memory block (with time-out).

**C format**

```c
ER tget_mpf (ID mpfid, VP *p_blk, TMO tmout);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mpfid; ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>O</td>
<td>VP</td>
<td>*p_blk; Start address of the acquired memory block.</td>
</tr>
<tr>
<td>I</td>
<td>TMO</td>
<td>tmout; Specified time-out (in millisecond).</td>
</tr>
</tbody>
</table>

**Note 1**

Invoking tasks are queued to the target fixed-size memory pool wait queue in the order defined at creating the fixed-sized memory pool. (FIFO order or current priority order).

**Note 2**

The contents of the block are undefined.

**Explanation**

This service call acquires the fixed-sized memory block from the fixed-sized memory pool specified by parameter `mpfid` and stores the start address in the area specified by parameter `p_blk`. If no fixed-size memory blocks could be acquired from the target fixed-size memory pool (no available fixed-size memory blocks exist) when this service call is issued, this service call does not acquire the fixed-size memory block but queues the invoking task to the target fixed-size memory pool wait queue and moves it from the RUNNING state to the WAITING state with time-out (fixed-size memory block acquisition wait state).

The WAITING state for a fixed-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Fixed-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing rel_mpf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>A fixed-sized memory block was returned to the target fixed-sized memory pool as a result of issuing irel_mpf.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The fixed-sized memory pool is reset as a result of issuing vrst_mpf.</td>
<td>E_RST</td>
</tr>
<tr>
<td>The time specified by <code>tmout</code> has elapsed.</td>
<td>E_TMOUT</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_mpf while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>
Note 3 The alignment number of memory block is 1. Please perform the following, in order to enlarge the alignment number of memory blocks.

- Specify the memory block size to a multiple of the desired alignment number at creating the fixed-sized memory pool.
- Make the start address of the fixed-sized memory pool area into the address of the desired alignment number.

Note 4 TMO_FEVR is specified for wait time tmout, processing equivalent to get_mpf will be executed. When TMO_POL is specified, processing equivalent to pget_mpf will be executed.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_blk == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid &gt; VTMAX_MPF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_blk has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by mpfid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mpf while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_mpf)</td>
</tr>
</tbody>
</table>
Outline
Release fixed-sized memory block.

C format

ER rel_mpf (ID mpfid, VP blk);
ER irel_mpf (ID mpfid, VP blk);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mpfid;</td>
<td>ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>VP blk;</td>
<td>Start address of the memory block to be released.</td>
</tr>
</tbody>
</table>

Explanation
This service call returns the fixed-sized memory block specified by parameter blk to the fixed-sized memory pool specified by parameter mpfid.
If a task is queued to the target fixed-sized memory pool wait queue when this service call is issued, fixed-sized memory block return processing is not performed but fixed-sized memory blocks are returned to the relevant task (first task of wait queue).
As a result, the relevant task is unlinked from the wait queue and is moved from the WAITING state (WAITING state for a fixed-sized memory block) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blk == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blk is illegal.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpfid &gt; VTMAX_MPF</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The irel_mpf was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The rel_mpf was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for rel_mpf)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by mpfid does not exist.</td>
</tr>
</tbody>
</table>
Outline

Reference fixed-sized memory pool state.

C format

```c
ER ref_mpf (ID mpfid, T_RMPF *pk_rmpf);
ER iref_mpf (ID mpfid, T_RMPF *pk_rmpf);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mpfid;</td>
<td>ID number of the fixed-sized memory pool.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMPF *pk_rmpf;</td>
<td>Pointer to the packet returning the fixed-sized memory pool state.</td>
</tr>
</tbody>
</table>

[Fixed-sized memory pool state packet: T_RMPF]

```c
typedef struct t_rmpf {
    ID wtskid;         /*Existence of waiting task*/
    UINT fblkcnt;        /*Number of free memory blocks*/
} T_RMPF;
```

Explanation

Stores fixed-sized memory pool state packet (ID number of the task at the head of the wait queue, number of free memory blocks, etc.) of the fixed-sized memory pool specified by parameter `mpfid` in the area specified by parameter `pk_rmpf`.

- `wtskid`
  Stores whether a task is queued to the fixed-size memory pool.
  - **TSK_NONE**: No applicable task
  - **Value**: ID number of the task at the head of the wait queue

- `fblkcnt`
  Stores the number of free memory blocks.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
</tbody>
</table>
  - **pk_rmpf == NULL**
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_ID   | -18   | Invalid ID number.  
|        |       |  - `mpfid` ≤ 0  
|        |       |  - `mpfid` > `VTMAX_MPF`  |
| E_CTX  | -25   | Context error.  
|        |       |  - This service call was issued in the CPU locked state.  
|        |       |  - This service call was issued in the status "PSW.IPL > kernel interrupt mask level".  
|        |       | Note When the iref_mpf is issued from task or the ref_mpf is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.  |
| E_MACV | -26   | Memory access violation. (only for ref_mpf)  
|        |       |  - The operand-write access to the area indicated by `pk_rmpf` has not been permitted to the invoking task.  |
| E_NOEXS| -42   | Non-existent object.  
|        |       |  - The fixed-sized memory pool specified by `mpfid` does not exist.  |
19.2.11 Memory pool management functions (variable-sized memory pools)

The following shows the service calls provided by the RI600PX as the memory pool management functions (variable-sized memory pools).

Table 19-15 Memory Pool Management Functions (Variable-Sized Memory Pools)

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cre_mpl</td>
<td>Create variable-sized memory pool</td>
<td>Task</td>
</tr>
<tr>
<td>acre_mpl</td>
<td>Create variable-sized memory pool (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_mpl</td>
<td>Delete variable-sized memory pool</td>
<td>Task</td>
</tr>
<tr>
<td>get_mpl</td>
<td>Acquire variable-sized memory block (waiting forever)</td>
<td>Task</td>
</tr>
<tr>
<td>pget_mpl</td>
<td>Acquire variable-sized memory block (polling)</td>
<td>Task</td>
</tr>
<tr>
<td>ipget_mpl</td>
<td>Acquire variable-sized memory block (polling)</td>
<td>Non-task</td>
</tr>
<tr>
<td>tget_mpl</td>
<td>Acquire variable-sized memory block (with time-out)</td>
<td>Task</td>
</tr>
<tr>
<td>rel_mpl</td>
<td>Release variable-sized memory block</td>
<td>Task</td>
</tr>
<tr>
<td>ref_mpl</td>
<td>Reference variable-sized memory pool state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_mpl</td>
<td>Reference variable-sized memory pool state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
Outline

Create variable-sized memory pool.

C format

```c
ER      cre_mpl (ID mplid, T_CMPL *pk_cmpl );
ER_ID   acre_mpl ( T_CMPL *pk_cmpl );
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>T_CMPL *pk_cmpl;</td>
<td>Pointer to the packet containing the variable-sized memory pool creation information.</td>
</tr>
</tbody>
</table>

[Variable-sized memory pool creation information packet : T_CMPL]

```c
typedef struct  t_cmpl {
    ATR    mplatr;   /*variable-sized memory pool attribute*/
    SIZE   mplsz;    /*Size of variable-sized memory pool area (in bytes)*/
    VP     mpl;      /*Start address of the variable-sized memory pool area*/
    UINT   maxblksz; /*maximum memory block size (in bytes)*/
} T_CMPL;
```

Explanation

This service call can be called from tasks that belong to Trusted Domain.
The `cre_mpl` creates a variable-sized memory pool with variable-sized memory pool ID indicated by `mplid` according to the content of `pk_cmpl`. The `acre_mpl` creates a variable-sized memory pool according to the content of `pk_cmpl`, and returns the created variable-sized memory pool ID.

1 ) Variable-sized memory pool attribute (`mplatr`)
   Only TA_TFIFO can be specified for `mplatr`.
   - TA_TFIFO ( = 0x0000)
     Task wait queue is managed in FIFO order.

2 ) Size of variable-sized memory pool area (`mplsz`), Start address of the variable-sized memory pool area (`mpl`)
   The application acquires `mplsz` bytes of variable-sized memory pool area and specifies the start address for `mpl`. The `mpl` must be 4-bytes boundary.

Note 1 The RI600PX is not concerned of anything of the access permission to the variable-sized memory pool area. To access to the memory block from task, the memory pool area must be in the memory object with appropriate permission.
Note, the RI600PX generates management tables in the memory pool area. If the management table is rewritten by allocation, correct system operation cannot be guaranteed.

Note 2 The μITRON4.0 specification defines the function that the kernel allocates variable-sized memory pool area when NULL is specified for mpl. But RI600PX does not support this function.

Note 3 The alignment number of memory block is 4.

3 ) Maximum memory block size (maxblksz)
Specify the maximum memory block size for maxblksz. Note, the maximum size of memory block that can be actually acquired might be larger than maxblksz. For details, refer to “9.3.1 Size of Variable-sized memory block”.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_mpl. (Created variable-sized memory pool ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_mpl.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplatr != TA_TFIFO</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_cmpl == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpls &lt; 24, mpls &gt; VTMAX_AREA_SIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- maxblksz == 0, maxblksz &gt; 0x0BFFFFF4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpls is too small to maxblksz.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpl + mpls &gt; 0x10000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpl is not 4-bytes boundary.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_mpl)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &gt; VTMAX_MPL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pk_cmpl has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOMEM</td>
<td>-33</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mpl == NULL</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_mpl)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_mpl)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by mplid exists.</td>
</tr>
</tbody>
</table>
del_mpl

Outline
Delete variable-sized memory pool.

C format
ER del_mpl (ID mplid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
</tbody>
</table>

Explanation
This service call can be called from tasks that belong to Trusted Domain. This service call deletes the variable-sized memory pool indicated by mplid. When there are waiting tasks for the target variable-sized memory pool by using get_mpl or tget_mpl, this service call cancels the WAITING state of the tasks and returns E_DLT as a return value of the get_mpl or tget_mpl.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &gt; VTMAX_MPL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by mplid does not exist.</td>
</tr>
</tbody>
</table>
get_mpl

Outline
Acquire variable-sized memory block (waiting forever).

C format
ER get_mpl (ID mplid, UINT blksz, VP *p_blk);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>UINT blksz;</td>
<td>Memory block size to be acquired (in bytes).</td>
</tr>
<tr>
<td>O</td>
<td>VP *p_blk;</td>
<td>Start address of the acquired memory block.</td>
</tr>
</tbody>
</table>

Explanation
This service call acquires a variable-size memory block of the size specified by parameter blksz from the variable-size memory pool specified by parameter mplid, and stores its start address into the area specified by parameter p_blk. If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory blocks but queues the invoking task to the target variable-size memory pool wait queue and moves it from the RUNNING state to the WAITING state (variable-size memory block acquisition wait state). The WAITING state for a variable-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Variable-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable-size memory block that satisfies the requested size was returned to the target variable-size memory pool as a result of issuing rel_mpl.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcibly released from waiting by following either.</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept ter_tsk while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- The time specified by tmout for tget_mpl has elapsed.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The variable-sized memory pool is reset as a result of issuing vrst_mpl.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>Forced release from waiting (accept del_mpl while waiting).</td>
<td>E_DLT</td>
</tr>
</tbody>
</table>

Note 1  For the size of the memory block, refer to “9.3.1 Size of Variable-sized memory block”.
Note 2  Invoking tasks are queued to the target variable-size memory pool wait queue in the FIFO order.
Note 3  The contents of the block are undefined.

Note 4  The alignment number of memory blocks changes with creation method of the variable-sized memory pool.

- Created by system configuration file
  The alignment number is 1. To enlarge the alignment number to 4, specify unique section to Section name assigned to the memory pool area (mpl_section) in Variable-sized Memory Pool Information (variable_memorypool[]) and locate the section to 4-bytes boundary address when linking.

- Created by cre_mpl or acre_mpl
  The alignment number is 4.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blksz == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blkz exceeds the maximum size that can be acquired.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_blk == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &lt;= 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &gt; VTMAX_MPL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_blk has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by mplid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mpl while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_mpl).</td>
</tr>
</tbody>
</table>
**Outline**

Acquire variable-sized memory block (polling).

**C format**

```c
ER    pget_mpl (ID mplid, UINT blksz, VP *p_blk);
ER    ipget_mpl (ID mplid, UINT blksz, VP *p_blk);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>UINT blksz;</td>
<td>Memory block size to be acquired (in bytes).</td>
</tr>
<tr>
<td>O</td>
<td>VP *p_blk;</td>
<td>Start address of the acquired memory block.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call acquires a variable-size memory block of the size specified by parameter `blksz` from the variable-size memory pool specified by parameter `mplid`, and stores its start address into the area specified by parameter `p_blk`.

If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory block but returns "E_TMOUT".

- **Note 1** For the size of the memory block, refer to "9.3.1 Size of Variable-sized memory block".
- **Note 2** The contents of the block are undefined.
- **Note 3** The alignment number of memory blocks changes with creation method of the variable-sized memory pool.
  - Created by system configuration file
    - The alignment number is 1. To enlarge the alignment number to 4, specify unique section to `Section name assigned to the memory pool area (mpl_section)` in `Variable-sized Memory Pool Information (variable_memorypool[])` and locate the section to 4-bytes boundary address when linking.
  - Created by `cre_mpl` or `acre_mpl`
    - The alignment number is 4.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_PAR  | -17   | Parameter error.  
- blk.sz == 0  
- blk.sz exceeds the maximum size that can be acquired.  
- p_blk == NULL |
| E_ID   | -18   | Invalid ID number.  
- mplid ≤ 0  
- mplid > VTMAX_MPL |
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.  
Note When the ipget_mpl is issued from task or the pget_mpl is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed. |
| E_MACV | -26   | Memory access violation. (only for pget_mpl)  
- Stack pointer points out of user stack for invoking task.  
- The operand-write access to the area indicated by p_blk has not been permitted to the invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The variable-sized memory pool specified by mplid does not exist. |
| E_TMOUT| -50   | Polling failure. |
Outline
Acquire variable-sized memory block (with time-out).

C format
```c
ER  tget_mpl (ID mplid, UINT blksz, VP *p_blk, TMO tmout);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>UINT blksz;</td>
<td>Memory block size to be acquired (in bytes).</td>
</tr>
<tr>
<td>O</td>
<td>VP *p_blk;</td>
<td>Start address of the acquired memory block.</td>
</tr>
</tbody>
</table>

TMO_FEVR: Waiting forever.
TMO_POL: Polling.

Explanation
This service call acquires a variable-size memory block of the size specified by parameter blksz from the variable-size memory pool specified by parameter mplid, and stores its start address into the area specified by parameter p_blk. If no variable-size memory blocks could be acquired from the target variable-size memory pool (no successive areas equivalent to the requested size were available) when this service call is issued, this service call does not acquire variable-size memory blocks but queues the invoking task to the target variable-size memory pool wait queue and moves it from the RUNNING state to the WAITING state with time-out (variable-size memory block acquisition wait state). The WAITING state for a variable-sized memory block is cancelled in the following cases.

<table>
<thead>
<tr>
<th>WAITING State for a Variable-sized Memory Block Cancel Operation</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable-size memory block that satisfies the requested size was returned to the target variable-size memory pool as a result of issuing rel_mpl.</td>
<td>E_OK</td>
</tr>
<tr>
<td>The task at the top of the transmission wait queue was forcibly released from waiting by following either.</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- Forced release from waiting (accept ter_tsk while waiting).</td>
<td>E_OK</td>
</tr>
<tr>
<td>- The time specified by tmout for tget_mpl has elapsed.</td>
<td>E_OK</td>
</tr>
<tr>
<td>Forced release from waiting (accept rel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>Forced release from waiting (accept irel_wai while waiting).</td>
<td>E_RLWAI</td>
</tr>
<tr>
<td>The variable-sized memory pool is reset as a result of issuing vrst_mpl.</td>
<td>EV_RST</td>
</tr>
<tr>
<td>The time specified by tmout has elapsed.</td>
<td>E_TMOUT</td>
</tr>
</tbody>
</table>
Note 1 For the size of the memory block, refer to “9.3.1 Size of Variable-sized memory block”.
Note 2 Invoking tasks are queued to the target variable-size memory pool wait queue in the FIFO order.
Note 3 The contents of the block are undefined.
Note 4 The alignment number of memory blocks changes with creation method of the variable-sized memory pool.
   - Created by system configuration file
     The alignment number is 1. To enlarge the alignment number to 4, specify unique section to Section name assigned to the memory pool area (mpl_section) in Variable-sized Memory Pool Information (variable_memorypool[]) and locate the section to 4-bytes boundary address when linking.
   - Created by cre mpl or acre mpl
     The alignment number is 4.
Note 5 TMO_FEVR is specified for wait time tmout, processing equivalent to get mpl will be executed. When TMO_POL is specified, processing equivalent to pget mpl will be executed.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blksz == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- blksz exceeds the maximum size that can be acquired.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_blk == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &lt; -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- tmout &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &lt;= 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &gt; VTMAX_MPL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the dispatching disabled state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_blk has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by mplid does not exist.</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>-49</td>
<td>Forced release from the WAITING state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept rel_wai/irel_wai while waiting.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>-50</td>
<td>Polling failure or specified time has elapsed.</td>
</tr>
<tr>
<td>E_DLT</td>
<td>-51</td>
<td>Waiting object deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accept del_mpl while waiting.</td>
</tr>
<tr>
<td>EV_RST</td>
<td>-127</td>
<td>Released from WAITING state by the object reset (vrst_mpl)</td>
</tr>
</tbody>
</table>
**rel_mpl**

**Outline**
Release variable-sized memory block.

**C format**
```c
ER rel_mpl (ID mplid, VP blk);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>I</td>
<td>VP blk;</td>
<td>Start address of memory block to be released.</td>
</tr>
</tbody>
</table>

**Explanation**
This service call returns the variable-sized memory block specified by parameter blk to the variable-sized memory pool specified by parameter mplid. After returning the variable-size memory blocks, these service calls check the tasks queued to the target variable-size memory pool wait queue from the top, and assigns the memory if the size of memory requested by the wait queue is available. This operation continues until no tasks queued to the wait queue remain or no memory space is available. As a result, the task that acquired the memory is unlinked from the queue and moved from the WAITING state (variable-size memory block acquisition wait state) to the READY state, or from the WAITING-SUSPENDED state to the SUSPENDED state.

**Note** The RI600PX do only simple error detection for blk. If blk is illegal and the error is not detected, the operation is not guaranteed after that.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_PAR | -17   | Parameter error.  
|       |       | - blk == NULL    
|       |       | - blk is illegal. |
| E_ID  | -18   | Invalid ID number.  
|       |       | - mplid ≤ 0       
<p>|       |       | - mplid &gt; VMAX_MPL |</p>
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by mplid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Reference variable-sized memory pool state.

C format
ER      refmpl (ID mplid, T_RMPL *pk_rmpl);
ER      irefmpl (ID mplid, T_RMPL *pk_rmpl);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMPL *pk_rmpl;</td>
<td>Pointer to the packet returning the variable-sized memory pool state.</td>
</tr>
</tbody>
</table>

[Variable-sized memory pool state packet: T_RMPL]

typedef struct t_rmpl {
   ID      wtskid;         /*Existence of waiting task*/
   SIZE    fmplsz;         /*Total size of free memory blocks*/
   UINT    fblksz;         /*Maximum memory block size available*/
} T_RMPL;

Explanation
These service calls store the detailed information (ID number of the task at the head of the wait queue, total size of free memory blocks, etc.) of the variable-size memory pool specified by parameter mplid into the area specified by parameter pk_rmpl.

- wtskid
  Stores whether a task is queued to the variable-size memory pool wait queue.
  
  TSK_NONE: No applicable task
  Value: ID number of the task at the head of the wait queue

- fmplsz
  Stores the total size of free memory blocks (in bytes).

- fblksz
  Stores the maximum memory block size available (in bytes).
### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>pk_rmpl</code> == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mplid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mplid</code> &gt; <code>VTMAX_MPL</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the iref_mpl is issued from task or the ref_mpl is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_mpl)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by <code>pk_rmpl</code> has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The variable-sized memory pool specified by <code>mplid</code> does not exist.</td>
</tr>
</tbody>
</table>
### 19.2.12 Time management functions

The following shows the service calls provided by the RI600PX as the time management functions.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_tim</td>
<td>Set system time</td>
<td>Task</td>
</tr>
<tr>
<td>iset_tim</td>
<td>Set system time</td>
<td>Non-task</td>
</tr>
<tr>
<td>get_tim</td>
<td>Reference system time</td>
<td>Task</td>
</tr>
<tr>
<td>iget_tim</td>
<td>Reference system time</td>
<td>Non-task</td>
</tr>
<tr>
<td>cre_cyc</td>
<td>Create cyclic handler</td>
<td>Task</td>
</tr>
<tr>
<td>acre_cyc</td>
<td>Create cyclic handler (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_cyc</td>
<td>Delete cyclic handler</td>
<td>Task</td>
</tr>
<tr>
<td>sta_cyc</td>
<td>Start cyclic handler operation</td>
<td>Task</td>
</tr>
<tr>
<td>ista_cyc</td>
<td>Start cyclic handler operation</td>
<td>Non-task</td>
</tr>
<tr>
<td>stp_cyc</td>
<td>Stop cyclic handler operation</td>
<td>Task</td>
</tr>
<tr>
<td>istp_cyc</td>
<td>Stop cyclic handler operation</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_cyc</td>
<td>Reference cyclic handler state</td>
<td>Task</td>
</tr>
<tr>
<td>iref_cyc</td>
<td>Reference cyclic handler state</td>
<td>Non-task</td>
</tr>
<tr>
<td>cre_alm</td>
<td>Create alarm handler</td>
<td>Task</td>
</tr>
<tr>
<td>acre_alm</td>
<td>Create alarm handler (automatic ID assignment)</td>
<td>Task</td>
</tr>
<tr>
<td>del_alm</td>
<td>Delete alarm handler</td>
<td>Task</td>
</tr>
<tr>
<td>sta_alm</td>
<td>Start alarm handler operation</td>
<td>Task</td>
</tr>
<tr>
<td>ista_alm</td>
<td>Start alarm handler operation</td>
<td>Non-task</td>
</tr>
<tr>
<td>stp_alm</td>
<td>Stop alarm handler operation</td>
<td>Task</td>
</tr>
<tr>
<td>istp_alm</td>
<td>Stop alarm handler operation</td>
<td>Non-task</td>
</tr>
<tr>
<td>ref_alm</td>
<td>Reference alarm handler state</td>
<td>Task</td>
</tr>
<tr>
<td>Irene_alm</td>
<td>Reference alarm handler state</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
set_tim
iset_tim

Outline
Set system time.

C format
ER set_tim (SYSTIM *p_systim);
ER iset_tim (SYSTIM *p_systim);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>SYSTIM *p_systim</td>
<td>Time to set as system time.</td>
</tr>
</tbody>
</table>

[System time packet: SYSTIM]

typedef struct systim {
    UH utime;          /*System time (higher 16 bits)*/
    UW ltime;          /*System time (lower 32 bits)*/
} SYSTIM;

Explanation
These service calls change the RI600PX system time (unit: msec) to the time specified by parameter p_systim.

Note Even if the system time is changed, the actual time at which the time management requests made before that (e.g., task time-outs, task delay by dly_tsk, cyclic handlers, and alarm handlers) are generated will not change.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_systim == NULL</td>
</tr>
</tbody>
</table>
### E_CTX -25

Context error.
- This service call was issued in the CPU locked state.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.

Note When the iset_tim is issued from task or the set_tim is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.

### E_MACV -26

Memory access violation. (only for set_tim)
- The operand-read access to the area indicated by `p_systim` has not been permitted to the invoking task.
Outline
Reference system time.

C format

```c
ER get_tim (SYSTIM *p_systim);
ER iget_tim (SYSTIM *p_systim);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>SYSTIM *p_systim;</td>
<td>Current system time.</td>
</tr>
</tbody>
</table>

[System time packet: SYSTIM]

```c
typedef struct systim {
    UH utime;          /*System time (higher 16 bits)*/
    UW ltime;          /*System time (lower 32 bits)*/
} SYSTIM;
```

Explanation
These service calls store the RI600PX system time (unit: msec) into the area specified by parameter p_systim.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_systim == NULL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
</tbody>
</table>

Note When the iget_tim is issued from task or the get_tim is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for get_tim)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_systim has not been</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permitted to the invoking task.</td>
</tr>
</tbody>
</table>
Outline
Create cyclic handler.

C format

ER cre_cyc (ID cycid, T_CCYC *pk_ccyc );
ER_ID acre_cyc ( T_CCYC *pk_ccyc );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID cycid;</td>
<td>ID number of the cyclic handler.</td>
</tr>
<tr>
<td>I</td>
<td>T_CCYC *pk_ccyc;</td>
<td>Pointer to the packet containing the cyclic handler creation information.</td>
</tr>
</tbody>
</table>

[Cyclic handler creation information packet : T_CCYC]

typedef struct t_ccyc {
    ATR cycatr; /*Cyclic handler attribute*/
    VP_INT exinf; /*Extended information*/
    FP cychdr; /*Cyclic handler start address*/
    RELTIM cyctim; /*Activation cycle (in milli-second)*/
    UINT cycphs; /*Activation phase (in milli-second)*/
} T_CCYC;

Explanation
This service call can be called from tasks that belong to Trusted Domain.
The cre_cyc creates a cyclic handler with cyclic handler ID indicated by cycid according to the content of pk_ccyc. The acre_cyc creates a cyclic handler according to the content of pk_ccyc, and returns the created cyclic handler ID.

1) Cyclic handler attribute (cycatr)
The following are specified for cycatr.

- TA_HLMG (= 0x0000)
  Only C-language is supported for cyclic handler description language.

- TA_STA (= 0x0002)
  When TA_STA is specified, the cyclic handler is in operational state (STA state). When TA_STA is not specified, the cyclic handler is in non-operational state (STP state).

- TA_PHS (= 0x0004)
  When TA_PHS is specified, the next activation time is determined preserving the activation phase when the cyclic handler is moved to operational state. When TA_PHS is not specified, the cyclic handler is activated cyctim
milliseconds after issuing sta_cyc or ista_cyc.  
Please refer to "10.6.5 Start cyclic handler operation".

2 ) Extended information (exinf)
   The exinf is passed to the cyclic handler as argument.  The exinf can be widely used by the user, for example, to set information concerning the cyclic handler.

3 ) Cyclic handler start address (cychdr)
   Specify the cyclic handler start address for cychdr.

4 ) Activation cycle (cyctim), activation phase (cycphs)
   Specify activation cycle (in milli-second) for cyctim.
   And specify the time to the first staring from this service call (in milli-second) for cycphs. When both TA_STA and TA_PHS are not specified, the cycphs is ignored.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive value</td>
<td>Normal completion of acre_cyc. (Created cyclic handler ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_cyc.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either of bits in cycatr except bit1 and bit2 is 1.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_ccyc == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cychdr == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cyctim == 0, cyctim &gt; (0x7FFFFFFF - TIC_NUME) / TIC_DENO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cyctim &lt; cycphs</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_cyc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid &gt; VTMAX_CYH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pk_ccyc has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_cyc)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_cyc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The cyclic handler specified by cycid exists.</td>
</tr>
</tbody>
</table>
**del_cyc**

**Outline**
Delete cyclic handler.

**C format**

```c
ER      del_cyc (ID cycid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID cycid</td>
<td>ID number of the cyclic handler.</td>
</tr>
</tbody>
</table>

**Explanation**
This service call can be called from tasks that belong to Trusted Domain. This service call deletes the cyclic handler indicated by cycid.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid &gt; VTMAX_CYH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The cyclic handler specified by cycid does not exist.</td>
</tr>
</tbody>
</table>

-Trust Domain.
Outline
Start cyclic handler operation.

C format
ER      sta_cyc (ID cycid);
ER      ista_cyc (ID cycid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID cycid</td>
<td>ID number of the cyclic handler.</td>
</tr>
</tbody>
</table>

Explanation
This service call moves the cyclic handler specified by parameter cycid from the non-operational state (STP state) to operational state (STA state).
As a result, the target cyclic handler is handled as an activation target of the RI600PX.
The relative interval from when either of this service call is issued until the first activation request is issued varies depending on whether the TA_PHS attribute is specified for the target cyclic handler at creation. For details, refer to "10.6.5 Start cyclic handler operation".

- When the TA_PHS attribute is specified
  The target cyclic handler activation timing is set up based on the activation phases and activation cycle.
  If the target cyclic handler has already been in operational state, however, no processing is performed even if this service call is issued, but it is not handled as an error.

- When the TA_PHS attribute is not specified
  The target cyclic handler activation timing is set up according to the activation cycle on the basis of the call time of this service call.
  This setting is performed regardless of the operating status of the target cyclic handler.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid &gt; VTMAX_CYH</td>
</tr>
</tbody>
</table>

E_CTX -25  
Context error.  
- This service call was issued in the CPU locked state.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.  

Note  When the ista_cyc is issued from task or the sta_cyc is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.

E_NOEXS -42  
Non-existent object.  
- The cyclic handler specified by cycid does not exist.
Outline
Stop cyclic handler operation.

C format

ER stp_cyc (ID cycid);
ER istp_cyc (ID cycid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID cycid</td>
<td>ID number of the cyclic handler.</td>
</tr>
</tbody>
</table>

Explanation
This service call moves the cyclic handler specified by parameter cycid from the operational state (STA state) to non-operational state (STP state). As a result, the target cyclic handler is excluded from activation targets of the RI600PX until issuance of sta_cyc or ista_cyc.

Note This service call does not perform queuing of stop requests. If the target cyclic handler has been moved to the non-operational state (STP state), therefore, no processing is performed but it is not handled as an error.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- cycid &gt; VTMAX_CYH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the istp_cyc is issued from task or the stp_cyc is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The cyclic handler specified by cycid does not exist.</td>
</tr>
</tbody>
</table>
ref_cyc
iref_cyc

Outline
Reference cyclic handler state.

C format

ER ref_cyc (ID cycid, T_RCYC *pk_rcyc);
ER iref_cyc (ID cycid, T_RCYC *pk_rcyc);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID cycid;</td>
<td>ID number of the cyclic handler.</td>
</tr>
<tr>
<td>O</td>
<td>T_RCYC *pk_rcyc;</td>
<td>Pointer to the packet returning the cyclic handler state.</td>
</tr>
</tbody>
</table>

[Cyclic handler state packet: T_RCYC]

typedef struct t_rcyc {
  STAT cycstat;        /*Current state*/
  RELTIM lefttim;      /*Time left before the next activation*/
} T_RCYC;

Explanation
Stores cyclic handler state packet (current state, time until the next activation, etc.) of the cyclic handler specified by parameter cycid in the area specified by parameter pk_rcyc.

- cycstat
  Store the current state.
  
  TCYC_STP: Non-operational state
  TCYC_STA: Operational state

- lefttim
  Stores the time until the next activation (in millisecond). When the target cyclic handler is in the non-operational state, lefttim is undefined.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_PAR | -17   | Parameter error.  
|       |       | - pk_rcyc == NULL |
| E_ID  | -18   | Invalid ID number.  
|       |       | - cycid ≤ 0  
|       |       | - cycid > VTMAX_CYH |
| E_CTX | -25   | Context error.  
|       |       | - This service call was issued in the CPU locked state.  
|       |       | - This service call was issued in the status "PSW.IPL > kernel interrupt mask level".  
|       |       | **Note** When the ref_cyc is issued from task or the ref_cyc is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed. |
| E_MACV| -26   | Memory access violation. (only for ref_cyc)  
|       |       | - The operand-write access to the area indicated by pk_rcyc has not been permitted to the invoking task. |
| E_NOEXS| -42  | Non-existent object.  
|       |       | - The cyclic handler specified by cycid does not exist. |
Create alarm handler.

C format

ER cre_alm (ID almid, T_CALM *pk_calm );
ER_ID acre_alm ( T_CALM *pk_calm );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID almid;</td>
<td>ID number of the alarm handler.</td>
</tr>
<tr>
<td>I</td>
<td>T_CALM *pk_calm;</td>
<td>Pointer to the packet containing the alarm handler creation information.</td>
</tr>
</tbody>
</table>

[Alarm handler creation information packet : T_CALM]

typedef struct t_calm {
    ATR     almatr;         /*Alarm handler attribute*/
    VP_INT  exinf;          /*Extended information*/
    FP      almhdr;         /*Alarm handler start address*/
} T_CALM;

Explanation

This service call can be called from tasks that belong to Trusted Domain. The cre_alm creates a alarm handler with alarm handler ID indicated by almid according to the content of pk_calm. The acre_alm creates a alarm handler according to the content of pk_calm, and returns the created alarm handler ID.

1 ) Alarm handler attribute (almatr)
   Only TA_HLNG can be specified for almatr.
   - TA_HLNG ( = 0x0000)
     Only C-language is supported for alarm handler description language.

2 ) Extended information (exinf)
   The exinf is passed to the alarm handler as argument. The exinf can be widely used by the user, for example, to set information concerning the alarm handler.

3 ) Alarm handler start address (almhdr)
   Specify the alarm handler start address for almhdr.
### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Positive value</td>
<td>Normal completion of acre_alm. (Created alarm handler ID)</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion of cre_alm.</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>-11</td>
<td>Reserved attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almattr != TA_HLNG</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_calm == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almhdr == NULL</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number. (only for cre_alm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid &gt; VTMAX_ALH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by pk_calm has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOID</td>
<td>-34</td>
<td>No ID number available. (only for acre_alm)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>-41</td>
<td>Object state error. (only for cre_alm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The alarm handler specified by almid exists.</td>
</tr>
</tbody>
</table>
**del_alm**

**Outline**
Delete alarm handler.

**C format**
```c
ER del_alm (ID almid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>almid;</td>
</tr>
</tbody>
</table>

**Explanation**
This service call can be called from tasks that belong to Trusted Domain.
This service call deletes the alarm handler indicated by almid.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid &gt; VTMAX_ALH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The alarm handler specified by almid does not exist.</td>
</tr>
</tbody>
</table>
### sta_alm

### ista_alm

#### Outline

Start alarm handler operation.

#### C format

```c
ER      sta_alm (ID almid, RELTIM almtim);
ER      ista_alm (ID almid, RELTIM almtim);
```

#### Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>ID number of the alarm handler.</td>
</tr>
<tr>
<td>I</td>
<td>RELTIM</td>
<td>Activation time (unit: msec)</td>
</tr>
</tbody>
</table>

#### Explanation

This service call sets to start the alarm handler specified by parameter `almid` in `almtim` msec and moves the target alarm handler from the non-operational state (STP state) to operational state (STA state). As a result, the target alarm handler is handled as an activation target of the RI600PX.

- **Note 1** When 0 is specified for `almtim`, the alarm handler will start at next base clock interrupt.
- **Note 2** This service call sets the activation time even if the target alarm handler has already been in the operational state. The previous activation time becomes invalid.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>almtim &gt; (0xFFFFFFF - TIC_NUME) / TIC_DENO</code></td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>almid ≤ 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>almid &gt; VTMAX_ALH</code></td>
</tr>
</tbody>
</table>
## Context Error

- This service call was issued in the CPU locked state.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.

**Note** When the ista alm is issued from task or the sta alm is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.

## Non-existent Object

- The alarm handler specified by almid does not exist.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> When the ista alm is issued from task or the sta alm is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The alarm handler specified by almid does not exist.</td>
</tr>
</tbody>
</table>
Outline
Stop alarm handler operation.

C format
ER stp_alm (ID almid);
ER istp_alm (ID almid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID almid</td>
<td>ID number of the alarm handler.</td>
</tr>
</tbody>
</table>

Explanation
This service call moves the alarm handler specified by parameter almid from the operational state (STA state) to non-operational state (STP state). As a result, the target alarm handler is excluded from activation targets of the RI600PX until issuance of sta_alm or ista_alm.

Note This service call does not perform queuing of stop requests. If the target alarm handler has been moved to the non-operational state (STP state), therefore, no processing is performed but it is not handled as an error.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- almid &gt; VTMAX_ALH</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note When the istp_alm is issued from task or the stp_alm is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The alarm handler specified by almid does not exist.</td>
</tr>
</tbody>
</table>
**Outline**

Reference alarm handler state.

**C format**

```c
ER      ref_alm (ID alm_id, T_RALM *pk_ralm);
ER      iref_alm (ID alm_id, T_RALM *pk_ralm);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID alm_id;</td>
<td>ID number of the alarm handler.</td>
</tr>
<tr>
<td>O</td>
<td>T_RALM *pk_ralm;</td>
<td>Pointer to the packet returning the alarm handler state.</td>
</tr>
</tbody>
</table>

[Alarm handler state packet: T_RALM]

```c
typedef struct  t_ralm {
    STAT almstat;        /*Current state*/
    RELTIM lefttim;        /*Time left before the next activation*/
} T_RALM;
```

**Explanation**

Stores alarm handler state packet (current state, time until the next activation, etc.) of the alarm handler specified by parameter alm_id in the area specified by parameter pk_ralm.

- **almstat**
  
  Store the current state.
  
  - **TALM_STP:** Non-operational state
  - **TALM_STA:** Operational state

- **lefttim**
  
  Stores the time until the next activation (in millisecond). When the target alarm handler is in the non-operational state, lefttim is undefined.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_PAR  | -17   | Parameter error.  
- \( pk\_ralm == \text{NULL} \) |
| E_ID   | -18   | Invalid ID number.  
- \( almid \leq 0 \)  
- \( almid > \text{VTMAX\_ALH} \) |
| E_CTX  | -25   | Context error.  
- This service call was issued in the CPU locked state.  
- This service call was issued in the status "PSW.IPL > kernel interrupt mask level".  
Note: When the iref\_alm is issued from task or the ref\_alm is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed. |
| E_MACV | -26   | Memory access violation. (only for ref\_alm)  
- The operand-write access to the area indicated by \( pk\_ralm \) has not been permitted to the invoking task. |
| E_NOEXS| -42   | Non-existent object.  
- The alarm handler specified by \( almid \) does not exist. |
### 19.2.13 System state management functions

The following shows the service calls provided by the RI600PX as the system state management functions.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>rot_rdq</td>
<td>Rotate task precedence</td>
<td>Task</td>
</tr>
<tr>
<td>irot_rdq</td>
<td>Rotate task precedence</td>
<td>Non-task</td>
</tr>
<tr>
<td>get_tid</td>
<td>Reference task ID in the RUNNING state</td>
<td>Task</td>
</tr>
<tr>
<td>iget_tid</td>
<td>Reference task ID in the RUNNING state</td>
<td>Non-task</td>
</tr>
<tr>
<td>loc_cpu</td>
<td>Lock the CPU</td>
<td>Task</td>
</tr>
<tr>
<td>iloc_cpu</td>
<td>Lock the CPU</td>
<td>Non-task</td>
</tr>
<tr>
<td>unl_cpu</td>
<td>Unlock the CPU</td>
<td>Task</td>
</tr>
<tr>
<td>iunl_cpu</td>
<td>Unlock the CPU</td>
<td>Non-task</td>
</tr>
<tr>
<td>dis_dsp</td>
<td>Disable dispatching</td>
<td>Task</td>
</tr>
<tr>
<td>ena_dsp</td>
<td>Enable dispatching</td>
<td>Task</td>
</tr>
<tr>
<td>sns_ctx</td>
<td>Reference contexts</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>sns_loc</td>
<td>Reference CPU locked state</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>sns_dsp</td>
<td>Reference dispatching disabled state</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>sns_dpn</td>
<td>Reference dispatch pending state</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>vsys_dwn</td>
<td>System down</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>ivsys_dwn</td>
<td>System down</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>vsta_knl</td>
<td>Start RI600PX</td>
<td>Task, Non-task</td>
</tr>
<tr>
<td>ivsta_knl</td>
<td>Start RI600PX</td>
<td>Task, Non-task</td>
</tr>
</tbody>
</table>
## Outline

Rotate task precedence.

## C format

```
ER  rot_rdq (PRI tskpri);
ER  irot_rdq (PRI tskpri);
```

## Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>PRI tskpri;</td>
<td>Priority of the tasks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPRI_SELF: Current priority of the invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: Priority of the tasks.</td>
</tr>
</tbody>
</table>

## Explanation

This service call re-queues the first task of the ready queue corresponding to the priority specified by parameter `tskpri` to the end of the queue to change the task execution order explicitly.

**Note 1** This service call does not perform queuing of rotation requests. If no task is queued to the ready queue corresponding to the relevant priority, therefore, no processing is performed but it is not handled as an error.

**Note 2** Round-robin scheduling can be implemented by issuing this service call via a cyclic handler in a constant cycle.

**Note 3** The ready queue is a hash table that uses priority as the key, and tasks that have entered an executable state (READY state or RUNNING state) are queued in FIFO order. Therefore, the scheduler realizes the RI600PX's scheduling system by executing task detection processing from the highest priority level of the ready queue upon activation, and upon detection of queued tasks, giving the CPU use right to the first task of the proper priority level.

**Note 4** As for a task which has locked mutexes, the current priority might be different from the base priority. In this case, even if the task issues this service call specifying `TPRI_SELF` for parameter `tskpri`, the ready queue of the current priority that the invoking task belongs cannot be changed.

**Note 5** For current priority and base priority, refer to "8.2.2 Current priority and base priority".

## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
### E_PAR -17
Parameter error.
- `tskpri < 0`
- `tskpri > TMAX_TPRI`
- When this service call was issued from a non-task, `TPRI_SELF` was specified `tskpri`.

### E_CTX -25
Context error.
- This service call was issued in the CPU locked state.
- The `irot_rdq` was issued from task.
- The `rot_rdq` was issued from non-task.
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.

### E_MACV -26
Memory access violation. (only for `rot_rdq`)
- Stack pointer points out of user stack for invoking task.
get_tid
iget_tid

Outline
Reference task ID in the RUNNING state.

C format
ER      get_tid (ID *p_tskid);
ER      iget_tid (ID *p_tskid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ID *p_tskid;</td>
<td>Pointer to the area returning the task ID number.</td>
</tr>
</tbody>
</table>

Explanation
These service calls store the ID of a task in the RUNNING state in the area specified by parameter p_tskid. This service call stores TSK_NONE in the area specified by parameter p_tskid if no tasks that have entered the RUNNING state exist.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_tskid == NULL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: When the iget_tid is issued from task or the get_tid is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for get_tid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_tskid has not been permitted to the invoking task.</td>
</tr>
</tbody>
</table>
loc_cpu
iloc_cpu

Outline
Lock the CPU.

C format

ER      loc_cpu (void);
ER      iloc_cpu (void);

Parameter(s)
None.

Explanation
These service calls transit the system to the CPU locked state.
In the CPU locked state, the task scheduling is prohibited, and kernel interrupts are masked. Therefore, exclusive processing can be achieved for all processing programs except non-kernel interrupt handlers.
The service calls that can be issued in the CPU locked state are limited to the one listed below.

<table>
<thead>
<tr>
<th>Service Call that can be issued</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_tsk</td>
<td>Terminate invoking task. (This service call transit the system to the CPU unlocked state.)</td>
</tr>
<tr>
<td>exd_tsk</td>
<td>Terminate and delete invoking task. (This service call transit the system to the CPU unlocked state.)</td>
</tr>
<tr>
<td>sns_tex</td>
<td>Reference task exception disabled state</td>
</tr>
<tr>
<td>loc_cpu, iloc_cpu</td>
<td>Lock the CPU.</td>
</tr>
<tr>
<td>unl_cpu, iunl_cpu</td>
<td>Unlock the CPU.</td>
</tr>
<tr>
<td>sns_loc</td>
<td>Reference CPU state.</td>
</tr>
<tr>
<td>sns_dsp</td>
<td>Reference dispatching state.</td>
</tr>
<tr>
<td>sns_ctx</td>
<td>Reference contexts.</td>
</tr>
<tr>
<td>sns_dpn</td>
<td>Reference dispatch pending state.</td>
</tr>
<tr>
<td>vsys_dwn, ivsys_dwn</td>
<td>System down</td>
</tr>
</tbody>
</table>

The unl_cpu, iunl_cpu ext_tsk and exd_tsk releases from the CPU locked state.

Note 1 The CPU locked state changed by issuing these service calls must be cancelled before the processing program that issued this service call ends.

Note 2 These service calls do not perform queuing of lock requests. If the system is in the CPU locked state, therefore, no processing is performed but it is not handled as an error.

Note 3 The RI600PX realizes the TIME MANAGEMENT FUNCTIONS by using base clock timer interrupts that occurs at constant intervals. If acknowledgment of the relevant base clock timer interrupt is disabled by issuing this service call, the TIME MANAGEMENT FUNCTIONS may no longer operate normally.
Note 4  For kernel interrupts, refer to **“12.1 Interrupt Type”**.

Note 5  The loc_cpu returns E_ILUSE error while interrupt mask has changed to other than 0 by `chg_ims`.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error. - This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”. Note When the iloc_cpu is issued from task or the loc_cpu is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_ILUSE</td>
<td>-28</td>
<td>Illegal use of service call. - This service call is issued in the status that the invoking task changes the PSW.IPL to other than 0 by using <code>chg_ims</code>.</td>
</tr>
</tbody>
</table>
### Outline

Unlock the CPU.

### C format

```c
ER      unl_cpu (void);
ER      iunl_cpu (void);
```

### Parameter(s)

None.

### Explanation

These service calls transit the system to the CPU unlocked state.

**Note 1** These service calls do not perform queuing of cancellation requests. If the system is in the CPU unlocked state, therefore, no processing is performed but it is not handled as an error.

**Note 2** These service calls do not cancel the dispatching disabled state that was set by issuing `dis_dsp`.

**Note 3** The CPU locked state is also cancelled by `ext_tsk` or `exd_tsk`.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The iunl_cpu was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The unl_cpu was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for unl_cpu)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
</tbody>
</table>
Outline
Disable dispatching.

C format

```c
ER dis_dsp (void);
```

Parameter(s)
None.

Explanation
This service call transits the system to the dispatching disabled state.
In the dispatching disabled state, the task scheduling is prohibited. Therefore, exclusive processing can be achieved for all tasks.
The operation that transit the system to the dispatching disabled state is as follows.

- `dis_dsp`
- `chg_ims` that changes PSW.IPL to other than 0.

The operation that transit the system to the dispatching enabled state is as follows.

- `ena_dsp`
- `ext_tsk`
- `exd_tsk`
- `chg_ims` that changes PSW.IPL to 0.

Note 1  The dispatching disabled state changed by issuing this service call must be cancelled before the task that issued this service call moves to the DORMANT state.

Note 2  This service call does not perform queuing of disable requests. If the system is in the dispatching disabled state, therefore, no processing is performed but it is not handled as an error.

Note 3  If a service call (such as `wai_sem`, `wai_flg`) that may move the status of the invoking task is issued while the dispatching disabled state, that service call returns `E_CTX` regardless of whether the required condition is immediately satisfied.
Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
</tbody>
</table>
### ena_dsp

#### Outline
Enable dispatching.

#### C format
```c
ER ena_dsp (void);
```

#### Parameter(s)
None.

#### Explanation
This service call transits the system to the dispatching enabled state.
The operation that changes in the dispatching disabled state is as follows.

- dis_dsp
- chg_ims that changes PSW.IPL to other than 0.

The operation that changes in the dispatching enabled state is as follows.

- ena_dsp
- ext_tsk
- exd_tsk
- chg_ims that changes PSW.IPL to 0.

**Note 1** This service call does not perform queuing of enable requests. If the system is in the dispatch enabled state, therefore, no processing is performed but it is not handled as an error.

**Note 2** If a service call (such as wai_sem, wai_flg) that may move the status of the invoking task is issued from when dis_dsp is issued until this service call is issued, the RI600PX returns E_CTX regardless of whether the required condition is immediately satisfied.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_CTX   | -25   | Context error.  
  - This service call was issued from a non-task.  
  - This service call was issued in the CPU locked state.  
  - This service call was issued in the status "PSW.IPL > kernel interrupt mask level". |
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
</tbody>
</table>
### sns_ctx

#### Outline
Reference contexts.

#### C format

```c
BOOL sns_ctx (void);
```

#### Parameter(s)
None.

#### Explanation
This service call examines the context type of the processing program that issues this service call. This service call returns TRUE when the processing program is non-task context, and return FALSE when the processing program is task context.

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
<td>Normal completion (non-task context).</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion (task context).</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
</tbody>
</table>
sns_loc

Outline
Reference CPU locked state.

C format

```c
BOOL sns_loc (void);
```

Parameter(s)
None.

Explanation
This service call examines whether the system is in the CPU locked state or not. This service call returns TRUE when the system is in the CPU locked state, and return FALSE when the system is in the CPU unlocked state.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
<td>Normal completion (CPU locked state).</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion (CPU unlocked state).</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
</tbody>
</table>
|        |       | - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”.

**sns_dsp**

**Outline**
Reference dispatching disabled state.

**C format**
```c
BOOL sns_dsp (void);
```

**Parameter(s)**
None.

**Explanation**
This service call examines whether the system is in the dispatching disabled state or not. This service call returns TRUE when the system is in the dispatching disabled state, and return FALSE when the system is in the dispatching enabled state.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
<td>Normal completion (dispatching disabled state).</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion (dispatching enabled state).</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
</tbody>
</table>
### Outline
Reference dispatch pending state.

### C format
```c
BOOL sns_dpn (void);
```

### Parameter(s)
None.

### Explanation
This service call examines whether the system is in the dispatch pending state or not. This service call returns TRUE when the system is in the dispatch pending state, and return FALSE when the system is not in the dispatch pending state. The state to fill either the following is called dispatch pending state.

- Dispatching disabled state
- CPU locked state
- PSW.IPL > 0, such as handlers

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
<td>Normal completion. (dispatch pending state)</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion. (any other states)</td>
</tr>
</tbody>
</table>
| E_CTX  | -25   | Context error.  
- This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
Outline
System down.

C format
void    vsys_dwn(W  type, VW  inf1, VW  inf2, VW  inf3);
void    vsys_dwn(W  type, VW  inf1, VW  inf2, VW  inf3);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>W  type;</td>
<td>Error type.</td>
</tr>
<tr>
<td>I</td>
<td>VW  inf1;</td>
<td>System down information 1</td>
</tr>
<tr>
<td>I</td>
<td>VW  inf2;</td>
<td>System down information 2</td>
</tr>
<tr>
<td>I</td>
<td>VW  inf3;</td>
<td>System down information 3</td>
</tr>
</tbody>
</table>

Explanation
These service calls pass the control to the System down routine (_RI_sys_dwn_()).
Specify the value (from 1 to 0x7FFFFFFF) typed to the occurring error for type. Note the value of 0 or less is reserved by the RI600PX.
These service calls never return.
For details of the parameter specification, refer to “15.2.2 Parameters of system down routine”.
These service calls are the function outside the range of μITRON4.0 specifications.

Note The system down routine is also called when abnormality is detected in the RI600PX.

Return value
None.
vsta_knl
ivsta_knl

Outline
Start RI600PX.

C format

```c
void vsta_knl( void );
void ivsta_knl( void );
```

Parameter(s)
None.

Explanation
These service start the RI600PX.
These service calls never return.
When these service call is issued, it is necessary to fill the following.

- All interrupts can not be accepted. (For example, PSW.I == 0)
- The CPU is in the supervisor mode (PSW.PM == 0).

The outline of processing of these service calls is shown as follows.

1 ) Initialize ISP register to the end address of SI section + 1
2 ) Initialize INTB register to the start address of the relocatable vector table (INTERRUPT_VECTOR section). The relocatable vector table is generated by the cfg600px.
3 ) Initialize the system time to 0.
4 ) Create various object which are defined in the system configuration file. If an error is detected in this process, the system will be down.
5 ) Initialize MPU (Memory Protection Unit).
6 ) Initialize base clock timer (call Base Clock Timer Initialization Routine (_RI_init_cmt_knl( )))
7 ) Pass control to scheduler

These service calls are the function outside the range of µITRON4.0 specifications.

Return value
None.
19.2.14 Interrupt management functions

The following shows the service calls provided by the RI600PX as the interrupt management functions.

Table 19-18 Interrupt Management Functions

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>chg_ims</td>
<td>Change interrupt mask</td>
<td>Task</td>
</tr>
<tr>
<td>ichg_ims</td>
<td>Change interrupt mask</td>
<td>Non-task</td>
</tr>
<tr>
<td>get_ims</td>
<td>Reference interrupt mask</td>
<td>Task</td>
</tr>
<tr>
<td>iget_ims</td>
<td>Reference interrupt mask</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
**Outline**

Change interrupt mask.

**C format**

```c
ER      chg_ims (IMASK imask);
ER      ichg_ims (IMASK imask);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>IMASK imask;</td>
<td>Interrupt mask desired.</td>
</tr>
</tbody>
</table>

**Explanation**

These service calls change PSW.IPL to the value specified by `imask`. Ranges of the value that can be specified for `imask` are from 0 to 15.

In the `chg_ims`, the system shifts to the dispatching disabled state when other than 0 is specified for `imask`, (it is equivalent to `dis_dsp`) and shifts to the dispatching enabled state when 0 is specified for `imask` (it is equivalent to `ena_dsp`).

On the other hand, the `ichg_ims` does not change the dispatching disabled / enabled state. The service calls that can be issued while PSW.IPL is larger than the Kernel interrupt mask level (`system_IPL`) are limited to the one listed below.

<table>
<thead>
<tr>
<th>Service Call that can be issued</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chg_ims, ichg_ims</code></td>
<td>Change interrupt mask.</td>
</tr>
<tr>
<td><code>get_ims, iget_ims</code></td>
<td>Reference interrupt mask</td>
</tr>
<tr>
<td><code>vsys_dwn, ivsys_dwn</code></td>
<td>System down</td>
</tr>
<tr>
<td><code>vsta_knl, ivsta_knl</code></td>
<td>Start RI600PX.</td>
</tr>
</tbody>
</table>

**Note 1** In the non-task, the interrupt mask must not lower PSW.IPL more than it starts.

**Note 2** The dispatching disabled state changed by issuing the `chg_ims` must be cancelled before the task that issued this service call moves to the DORMANT state.

**Note 3** If a service call (such as `wai_sem, wai_flg`) that may move the status of the invoking task is issued while the dispatching disabled state, that service call returns E_CTX regardless of whether the required condition is immediately satisfied.

**Note 4** The RI600PX realizes the TIME MANAGEMENT FUNCTIONS by using base clock timer interrupts that occurs at constant intervals. If acknowledgment of the relevant base clock timer interrupt is disabled by issuing this service call, the TIME MANAGEMENT FUNCTIONS may no longer operate normally.

**Note 5** Do not issue `ena_dsp` while a task changes PSW.IPL to other than 0 by using `chg_ims`. If issuing `ena_dsp`, the system moves to the dispatching enabled state. If task dispatching occurs, PSW is changed for the dispatched task. Therefore PSW.IPL may be lowered without intending it.

**Note 6** Refer to “12.8 Prohibit Maskable Interrupts”.

---

**Outline**

Change interrupt mask.

**C format**

```c
ER      chg_ims (IMASK imask);
ER      ichg_ims (IMASK imask);
```
## Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- imask &gt; 15</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The ichg_ims was issued from task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The chg_ims was issued from non-task.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for chg_ims)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
</tbody>
</table>
get_ims  
iget_ims

Outline
Reference interrupt mask.

C format
ER      get_ims (IMASK *p_imask);
ER      iget_ims (IMASK *p_imask);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>IMASK *p_imask;</td>
<td>Pointer to the area returning the interrupt mask.</td>
</tr>
</tbody>
</table>

Explanation
These service calls store PSW.IPL into the area specified by parameter p_imask.

Note 1  These service call do not detect the context error.

Note 2  The following intrinsic functions provided by compiler are higher-speed than this service call. See "CubeSuite+  

- get_ipl() : Refers to the interrupt priority level.
- get_psw() : Refers to PSW value.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_imask == NULL</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for get_ims)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by p_imask has not been permitted to the invoking task.</td>
</tr>
</tbody>
</table>
19.2.15 System configuration management functions

The following shows the service calls provided by the RI600PX as the system configuration management functions.

Table 19-19 System Configuration Management Functions

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref_ver</td>
<td>Reference version information</td>
<td>Task</td>
</tr>
<tr>
<td>iref_ver</td>
<td>Reference version information</td>
<td>Non-task</td>
</tr>
</tbody>
</table>
ref_ver
iref_ver

Outline
Reference version information.

C format
ER ref_ver (T_RVER *pk_rver);
ER iref_ver (T_RVER *pk_rver);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>T_RVER *pk_rver;</td>
<td>Pointer to the packet returning the version information.</td>
</tr>
</tbody>
</table>

[Version information packet: T_RVER]

typedef struct t_rver {
  UH maker;          /*Kernel maker code*/
  UH prid;           /*Identification number of the kernel*/
  UH spver;          /*Version number of the ITRON specification*/
  UH prver;          /*Version number of the kernel*/
  UH prno[4];        /*Management information of the kernel*/
} T_RVER;

Explanation
These service calls store the RI600PX version information into the area specified by parameter pk_rver.

- **maker**
The *maker* represents the manufacturer who created this kernel. In the RI600PX, 0x011B, which is the maker code assigned for Renesas Electronics Corporation, is returned for *maker*. Note, the value defined in the kernel configuration macro **TKERNEL_MAKER** is same as *maker*.

- **prid**
The *prid* represents the number that identifies the kernel and VLSI. In the RI600PX, 0x0004 is returned for *prid*. Note, the value defined in the kernel configuration macro **TKERNEL_PRID** is same as *prid*.

- **spver**
The *spver* represents the specification to which this kernel conforms. In the RI600PX, 0x5403 is returned for *spver*. Note, the value defined in the kernel configuration macro **TKERNEL_SPVER** is same as *spver*.

- **prver**
The *prver* represents the version number of this kernel. For example, 0x0123 is returned for *prver* when the kernel version is “V1.02.03”. Note, the value defined in the kernel configuration macro **TKERNEL_PRVER** is same as *prver*. 
- pmo
The pmo represents product management information and product number, etc. In the RI600PX, 0x0000 is returned for all pmos.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_rver == NULL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>Note</td>
<td></td>
<td>When the iref_ver is issued from task or the ref_ver is issued from non-task, the context error is not detected and normal operation of the system is not guaranteed.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation. (only for ref_ver)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rver has not been permitted to the invoking task.</td>
</tr>
</tbody>
</table>
19.2.16 Object reset functions

The following shows the service calls provided by the RI600PX as the object reset functions.

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>vrst_dtq</td>
<td>Reset data queue</td>
<td>Task</td>
</tr>
<tr>
<td>vrst_mbx</td>
<td>Reset mailbox</td>
<td>Task</td>
</tr>
<tr>
<td>vrst_mbf</td>
<td>Reset message buffer</td>
<td>Task</td>
</tr>
<tr>
<td>vrst_mpf</td>
<td>Reset fixed-sized memory pool</td>
<td>Task</td>
</tr>
<tr>
<td>vrst_mpl</td>
<td>Reset variable-sized memory pool</td>
<td>Task</td>
</tr>
</tbody>
</table>


### vrst_dtq

**Outline**
Reset data queue.

**C format**

```c
ER vrst_dtq (ID dtqid);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>dtqid; ID number of the data queue.</td>
</tr>
</tbody>
</table>

**Explanation**

This service call resets the data queue specified by parameter `dtqid`. The data having been accumulated by the data queue area are annulled. The tasks to wait to send data to the target data queue are released from the WAITING state, and `EV_RST` is returned as a return value for the tasks.

**Note 1** In this service call, the tasks to wait to receive data do not released from the WAITING state.

**Note 2** This service call is the function outside μITRON4.0 specification.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid</code> ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>dtqid</code> &gt; <code>VTMAX_DTQ</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The data queue specified by <code>dtqid</code> does not exist.</td>
</tr>
</tbody>
</table>
Outline

Reset mailbox.

C format

```
ER vrst_mbx (ID mbxid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mbxid; ID number of the mailbox.</td>
</tr>
</tbody>
</table>

Explanation

This service call reset the mailbox specified by parameter mbxid.
The messages having been accumulated by the mailbox come off from the management of the RI600PX.

Note 1  In this service call, the tasks to wait to receive message do not released from the WAITING state.
Note 2  This service call is the function outside μITRON4.0 specification.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbxid &gt; VTMAX_MBX</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The mailbox specified by mbxid does not exist.</td>
</tr>
</tbody>
</table>
vrst_mbf

Outline
Reset message buffer.

C format
ER vrst_mbf (ID mbfid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mbfid;</td>
<td>ID number of the message buffer.</td>
</tr>
</tbody>
</table>

Explanation
This service call reset the message buffer specified by parameter mbfid.
The messages having been accumulated by the message buffer area are annulled. The tasks to wait to send message to
the target message buffer are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

Note 1 In this service call, the tasks to wait to receive message do not released from the WAITING state.
Note 2 This service call is the function outside μITRON4.0 specification.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbfid &gt; VTMAX_MBF</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The message buffer specified by mbfid does not exist.</td>
</tr>
</tbody>
</table>
vrst_mpf

Outline
Reset fixed-sized memory pool.

C format

```c
ER    vrst_mpf (ID mpfid);
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID</td>
<td>mpfid; ID number of the fixed-sized memory pool.</td>
</tr>
</tbody>
</table>

Explanation

This service call resets the fixed-sized memory pool specified by parameter `mpfid`. The tasks to wait to get memory block from the target fixed-sized memory pool are released from the WAITING state, and `EV_RST` is returned as a return value for the tasks.

Note 1 All fixed-sized memory blocks that had already been acquired are returned to the target fixed-sized memory pool. Therefore, do not access those fixed-sized memory blocks after issuing this service call.

Note 2 This service call is the function outside μITRON4.0 specification.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mpfid ≤ 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>mpfid &gt; VTMAX_MPF</code></td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by <code>mpfid</code> does not exist.</td>
</tr>
</tbody>
</table>
Outline
Reset variable-sized memory pool.

C format
ER vrst_mpl (ID mplid);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID mplid;</td>
<td>ID number of the variable-sized memory pool.</td>
</tr>
</tbody>
</table>

Explanation
This service call reset the variable-sized memory pool specified by parameter mplid. The tasks to wait to get memory block from the target variable-sized memory pool are released from the WAITING state, and EV_RST is returned as a return value for the tasks.

Note 1 All variable-sized memory blocks that had already been acquired are returned to the target variable-sized memory pool. Therefore, do not access those variable-sized memory blocks after issuing this service call.

Note 2 This service call is the function outside µITRON4.0 specification.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_ID</td>
<td>-18</td>
<td>Invalid ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid ≤ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mplid &gt; VTMAX_MPL</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from a non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The fixed-sized memory pool specified by mpfid does not exist.</td>
</tr>
</tbody>
</table>
19.2.17 Memory object management functions

The following shows the service calls provided by the RI600PX as the memory object management functions.

Table 19-21 Memory Object Management Functions

<table>
<thead>
<tr>
<th>Service Call</th>
<th>Function</th>
<th>Useful Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ata_mem</td>
<td>Register memory object</td>
<td>Task</td>
</tr>
<tr>
<td>det_mem</td>
<td>Unregister memory object</td>
<td>Task</td>
</tr>
<tr>
<td>sac_mem</td>
<td>Change access permission vector for memory object</td>
<td>Task</td>
</tr>
<tr>
<td>vprb_mem</td>
<td>Check access permission</td>
<td>Task</td>
</tr>
<tr>
<td>ref_mem</td>
<td>Reference memory object state</td>
<td>Task</td>
</tr>
</tbody>
</table>
ata_mem

Outline
Register memory object.

C format

```c
ER ata_mem ( T_AMEM *pk_amem, ACVCT *p_acvct );
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>T_AMEM *pk_amem;</td>
<td>Pointer to the packet containing the memory object registration information.</td>
</tr>
<tr>
<td>I</td>
<td>ACVCT *p_acvct;</td>
<td>Pointer to the packet containing the access permission vector.</td>
</tr>
</tbody>
</table>

[Memory object registration information packet : T_AMEM]

```c
typedef struct t_amem {
    ATR mematr;         /*Memory object attribute*/
    VP base;            /*Memory object start address*/
    SIZE size;          /*Size of memory object (in bytes)*/
} T_AMEM;
```

[Access permission vector : ACVCT]

```c
typedef struct acvct {
    ACPTN acptn1;       /*Access permission pattern for operand-read*/
    ACPTN acptn2;       /*Access permission pattern for operand-write*/
    ACPTN acptn3;       /*Access permission pattern for execution*/
} ACVCT;
```

Explanation
This service call can be called from tasks that belong to Trusted Domain. This service call registers the area started from the address specified by base with the size [bytes] as the memory object with the access permission vector specified by p_acvct.
The bit N-1 in the access permission pattern shows whether tasks belonging to the domain ID #N can access the memory object. The bit value 1 means "permitted" and 0 means "not permitted".
The specified memory object area must satisfy the following.

A ) The start address (base) must be 16-bytes boundary. If not, this service call returns E_PAR error.
B ) The size (size) must be multiple of 16. If not, this service call returns E_PAR error.
C ) The memory object area must not either with all user stacks and all other memory objects. If not, this service call does not detect an error, and correct system operation cannot be guaranteed.
Note 1  The following macros are prepared to specify access permission vector.

- **TACT_SRW**
  Returns access permission vector that represents “all types of access (operand-read, operand-write, execution) are permitted for all domains”. This macro can be describe only at the right of an initial assignment statement.

- **TACT_SRO**
  Returns access permission vector that represents “operand-read access is permitted for all domains, and operand-write access and execution access are not permitted for all domains”. This macro can be describe only at the right of an initial assignment statement.

- **ACVCT TACT_PRW ( ID domid )**
  Returns access permission vector that represents “all types of access (operand-read, operand-write, execution) are permitted only for the domain indicated by domid”. This macro can be describe only at the right of an initial assignment statement.

- **ACVCT TACT_PRO ( ID domid )**
  Returns access permission vector that represents “operand-write access is not permitted for all domain, operand-read and execution access are permitted only for the domain indicated by domid”. This macro can be describe only at the right of an initial assignment statement.

- **ACVCT TACT_SRPW ( ID domid )**
  Returns access permission vector that represents “operand-read and execution access are permitted for all domain, operand-write access is permitted only for the domain indicated by domid”. This macro can be describe only at the right of an initial assignment statement.

Note 2  The following macros are prepared to specify access permission pattern.

- **TACP_SHARED**
  Returns access permission pattern that represents “all domain can access”.

- **ACPTN TACP ( ID domid )**
  Returns access permission pattern that represents “only the domain indicated by domid can access”.

Note 3  The memory object attribute (mematr) is merely ignored.

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pk_amem == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- base is not 16-bytes boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- size is not multiple of 16.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p_acvct == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- acptn1 == acptn2 == acptn3 == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either of bits corresponding to the domain ID that is larger than the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maximum domain ID (VTMAX_DOMAIN) of either acptn1, acptn2 or acptn3 is 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- size == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- base + size &gt; 0x1000000000</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E_MACV  | -26   | Memory access violation.  
- Stack pointer points out of user stack for invoking task.  
- The operand-read access to the area indicated by pk_amem has not been permitted to the invoking task.  
- The operand-read access to the area indicated by p_acvct has not been permitted to the invoking task. |
| E_OACV  | -27   | Object access violation.  
- The invoking task does not belong to trusted domain.  
- The number of memory objects from which the access is permitted to one domain exceeds 7. |
| E_OBJ   | -41   | Object state error.  
- The memory object started from the address specified by base has already been registered. |
**det_mem**

**Outline**
Unregister memory object.

**C format**
```
ER  det_mem (VP base);
```

**Parameter(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>VP</td>
<td>base;</td>
</tr>
</tbody>
</table>

**Explanation**
This service call can be called from tasks that belong to Trusted Domain.
This service call unregisters the memory object started from the address specified by base.

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
| E_CTX  | -25   | Context error.                  
  - This service call was issued in the CPU locked state. 
  - This service call was issued from non-task. 
  - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV | -26   | Memory access violation.          
  - Stack pointer points out of user stack for invoking task. |
| E_OACV | -27   | Object access violation.          
  - The invoking task does not belong to trusted domain. |
| E_NOEXS| -42   | Non-existent object.       
  - The memory object started from the address specified by base has already been registered. |
sac_mem

Outline
Change access permission vector for memory object.

C format

```c
ER sac_mem ( VP base, ACVCT *p_acvct );
```

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>VP base;</td>
<td>Memory object start address.</td>
</tr>
<tr>
<td>I</td>
<td>ACVCT *p_acvct;</td>
<td>Pointer to the packet containing the access permission vector.</td>
</tr>
</tbody>
</table>

[Access permission vector : ACVCT]

```c
typedef struct acvct {
    ACPTN acptn1;         /*Access permission pattern for operand-read*/
    ACPTN acptn2;         /*Access permission pattern for operand-write*/
    ACPTN acptn3;         /*Access permission pattern for execution*/
} ACVCT;
```

Explanation
This service call can be called from tasks that belong to Trusted Domain.
This service call changes the access permission vector for the memory object started from the address specified by `base` to the content indicated by `p_acvct`.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>Parameter error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>p_acvct</code> == NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>acptn1</code> == <code>acptn2</code> == <code>acptn3</code> == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Either of bits corresponding to the domain ID that is larger than the maximum domain ID (VTMAX_DOMAIN) of either <code>acptn1</code>, <code>acptn2</code> or <code>acptn3</code> is 1.</td>
</tr>
<tr>
<td>Macro</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status &quot;PSW.IPL &gt; kernel interrupt mask level&quot;.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-read access to the area indicated by p_acvc has not been permitted to the invoking task.</td>
</tr>
<tr>
<td>E_OACV</td>
<td>-27</td>
<td>Object access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The invoking task does not belong to trusted domain.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The number of memory objects from which the access is permitted to one domain exceeds 7.</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The memory object started from the address specified by base has already been registered.</td>
</tr>
</tbody>
</table>
vprb_mem

Outline
Check access permission.

C format
ER_BOOL vprb_mem ( VP base, SIZE size, ID tskid, MODE pmmode );

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>VP base;</td>
<td>Start address for checking</td>
</tr>
<tr>
<td>I</td>
<td>SIZE size;</td>
<td>Size of checking area (in bytes).</td>
</tr>
<tr>
<td>I</td>
<td>ID tskid;</td>
<td>ID number of the task.</td>
</tr>
<tr>
<td>I</td>
<td>PMMODE pmmode;</td>
<td>Access mode.</td>
</tr>
</tbody>
</table>

Explanation
This service call checks whether the task indicated by tskid has the access permission indicated by pmmode for the memory area of size bytes from the address specified by base. This service call returns TRUE when the access is permitted and returns FALSE when the access is not permitted. The following are specified for pmmode.

pmmode := ( TPM_READ | TPM_WRITE | TPM_EXEC )

- TPM_READ (= 0x0001)
  Checks whether operand-read access is permitted.

- TPM_WRITE (= 0x0002)
  Checks whether operand-write access is permitted.

- TPM_EXEC (= 0x0004)
  Checks whether execution access is permitted.

Note  This service call is the function outside μITRON4.0 specification.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>0</td>
<td>Normal completion. (The access is permitted.)</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>Normal completion.(The access is not permitted.)</td>
</tr>
</tbody>
</table>
### Macro Values Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| E_PAR  | -17   | Parameter error.  
          - `size == 0`  
          - `pmmode == 0`, One of bits except bit0, bit1 and bit2 of `pmmode` is 1. |
| E_ID   | -18   | Invalid ID number.  
          - `tskid < 0`  
          - `tskid > VMAX_TSK` |
| E_CTX  | -25   | Context error.  
          - This service call was issued in the CPU locked state.  
          - This service call was issued from non-task.  
          - This service call was issued in the status “PSW.IPL > kernel interrupt mask level”. |
| E_MACV | -26   | Memory access violation.  
          - Stack pointer points out of user stack for invoking task. |
| E_NOEXS| -42   | Non-existent object.  
          - The task specified by `tskid` does not exist. |
ref_mem

Outline
Reference memory object state.

C format
ER ref_mem (VP base, T_RMEM *pk_rmem);

Parameter(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>VP base;</td>
<td>Memory object start address.</td>
</tr>
<tr>
<td>O</td>
<td>T_RMEM *pk_rmem;</td>
<td>Pointer to the packet returning the mutex state.</td>
</tr>
</tbody>
</table>

[Access permission vector: ACVCT]

typedef struct acvct {
    ACPTN acptn1;         /*Access permission pattern for operand-read*/
    ACPTN acptn2;         /*Access permission pattern for operand-write*/
    ACPTN acptn3;         /*Access permission pattern for execution*/
} ACVCT;

[Memory object state packet: T_RMEM]

typedef struct t_rmem {
    ACVCT acvct;          /*Access permission vector*/
} T_RMEM;

Explanation
This service call stores the information of the memory object started from the address specified by parameter base into the area specified by parameter pk_rmem.
The bit N-1 in the access permission pattern shows whether tasks belonging to the domain ID #N can access the memory object. The bit value 1 means “permitted” and 0 means “not permitted”.

Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>E_PAR</td>
<td>-17</td>
<td>parameter error.</td>
</tr>
</tbody>
</table>
<pre><code>|       | - pk_rmem == NULL                                |
</code></pre>
<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>-25</td>
<td>Context error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the CPU locked state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued from non-task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- This service call was issued in the status “PSW.IPL &gt; kernel interrupt mask level”.</td>
</tr>
<tr>
<td>E_MACV</td>
<td>-26</td>
<td>Memory access violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stack pointer points out of user stack for invoking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The operand-write access to the area indicated by pk_rmem has not been</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permitted to the invoking task</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>-42</td>
<td>Non-existent object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The memory object started from the address specified by base has already</td>
</tr>
<tr>
<td></td>
<td></td>
<td>been registered.</td>
</tr>
</tbody>
</table>
CHAPTER 20 SYSTEM CONFIGURATION FILE

This chapter explains the coding method of the system configuration file required to output information files that contain data to be provided for the RI600PX.

20.1 Outline

The following shows the notation method of system configuration files.

- **Comment**
  Parts from two successive slashes (//) to the line end are regarded as comments.

- **Numeric**
  A numeric value can be written in one of the following formats. Note, do not specify the value exceeding 0xFFFFFFFF.
  
  **Hexadecimal:** Add “0x” or “0X” at the beginning of a numeric value or add “h” or “H” at the end. In the latter format, be sure to add “0” at the beginning when the value begins with an alphabetic letter from A to F or a to f. Note that the configurator does not distinguish between uppercase and lowercase letters for alphabetic letters (A to F or a to f) used in numeric value representation.
  
  **Decimal:** Simply write an integer value as is usually done (23, for example). Note that a decimal value must not begin with “0”.
  
  **Octal:** Add “0” at the beginning of a numeric value or add “O” or “o” at the end.
  
  **Binary:** Add “B” or “b” at the end of a numeric value. Note that a binary value must not begin with “0”.

- **Operator**
  The following operator can be used for numeric value.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
<th>Direction of Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>High</td>
<td>Left to right</td>
</tr>
<tr>
<td>- (unary minus)</td>
<td></td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td></td>
<td>Left to right</td>
</tr>
<tr>
<td>+ - (binary minus)</td>
<td>Low</td>
<td>Left to right</td>
</tr>
</tbody>
</table>

- **Symbol**
  A symbol is a string of numeric characters, uppercase alphabetic letters, lowercase alphabetic letters, and underscores (_). It must not begin with a numeric character.

- **Function name**
  A function name consists of numeric characters, uppercase alphabetic letters, lowercase alphabetic letters, underscores (_), and dollar signs ($). It must not begin with a numeric character and must end with “()”.
  
  To specify module name written by assembly language, name the module starting in '_', and specify the name that excludes '_' for function name.

- **Frequency**
  The frequency is indicated by a character string that consist of numerals and . (period), and ends with “MHz”. The numerical values are significant up to six decimal places. Also note that the frequency can be entered using
20.2 Default System Configuration File

For most definition items, if the user omits settings, the settings in the default system configuration file are used. The default system configuration file is stored in the folder indicated by environment variable “LIB600”. Be sure not to edit this file.

20.3 Configuration Information (static API)

The configuration information that is described in a system configuration file is shown as follows.

- System Information (system)
- Base Clock Interrupt Information (clock)
- Maximum ID (maxdefine)
- Domain Definition (domain[])
- Memory Object Definition (memory_object[])
- Task Information (task[])
- Semaphore Information (semaphore[])
- Eventflag Information (flag[])
- Data Queue Information (dataqueue[])
- Mailbox Information (mailbox[])
- Mutex Information (mutex[])
- Message Buffer Information (message_buffer[])
- Fixed-sized Memory Pool Information (memorypool[])
- Variable-sized Memory Pool Information (variable_memorypool[])
- Cyclic Handler Information (cyclic_hand[])
- Alarm Handler Information (alarm_hand[])
- Relocatable Vector Information (interrupt_vector[])
- Fixed Vector/Exception Vector Information (interrupt_fvector[])
20.4 System Information (system)

Here, information on the system whole is defined.
Only one “system” can be defined. And the “system” can not be omitted.

Format

Parentheses < > show the user input part.

system {
  stack_size  = <1. System stack size (stack_size)>;
  priority    = <2. Maximum task priority (priority)>;
  system_IPL  = <3. Kernel interrupt mask level (system_IPL)>;
  message_pri = <4. Maximum message priority (message_pri)>;
  tic_deno    = <5. Denominator of base clock interval time (tic_deno)>;
  tic_nume    = <6. Numerator of base clock interval time (tic_nume)>;
  context     = <7. Task context register (context)>;
};

1 ) System stack size (stack_size)
- Description
  Define the total stack size used in service call processing and interrupt processing.
- Definition format
  Numeric value
- Definition range
  More than 8, and multiple of 4.
- When omitting
  The set value in the default system configuration file (factory setting: 0x800) applied.

2 ) Maximum task priority (priority)
- Description
  Define the maximum task priority.
- Definition format
  Numeric value
- Definition range
  1 - 255
- When omitting
  The set value in the default system configuration file (factory setting: 32) applied.
  - TMAX_TPRI
    The cfg600px outputs the macro TMAX_TPRI which defines this setting to the system information header file "kernel_id.h".

3 ) Kernel interrupt mask level (system_IPL)
- Description
  Define the interrupt mask level when the kernel's critical section is executed (PSW register's IPL value).
  Interrupts with higher priority levels than that are handled as “non-kernel interrupts”.
  For details of “non-kernel interrupts” and “kernel interrupts”, refer to "12.1 Interrupt Type".
- Definition format
  Numeric value
- Definition range
  1 - 15
- When omitting
  The set value in the default system configuration file (factory setting: 7) applied.
- **VTKNL_LVL**
  The cfg600px outputs the macro `VTKNL_LVL` which defines this setting to the system information header file "kernel_id.h".

4 ) **Maximum message priority (message_pri)**

- **Description**
  Define the maximum message priority used in the mailbox function. Note that if the mailbox function is not used, this definition item has no effect.

- **Definition format**
  Numeric value

- **Definition range**
  1 - 255

- **When omitting**
  The set value in the default system configuration file (factory setting: 255) applied.

- **TMAX_MPRI**
  The cfg600px outputs the macro `TMAX_MPRI` which defines this setting to the system information header file "kernel_id.h".

5 ) **Denominator of base clock interval time (tic_deno)**

- **Description**
  The base clock interval time is calculated by the following expression. Either `tic_deno` or `tic_nume` should be 1.

  \[
  \text{The base clock interval time (in millisecond)} = \frac{\text{tic_nume}}{\text{tic_deno}}
  \]

- **Definition format**
  Numeric value

- **Definition range**
  1 - 100

- **When omitting**
  The set value in the default system configuration file (factory setting: 1) applied.

- **TIC_DENO**
  The cfg600px outputs the macro `TIC_DENO` which defines this setting to the system information header file "kernel_id.h".

6 ) **Numerator of base clock interval time (tic_nume)**

- **Description**
  See above.

- **Definition format**
  Numeric value

- **Definition range**
  1 - 65535

- **When omitting**
  The set value in the default system configuration file (factory setting: 1) applied.

- **TIC_NUME**
  The cfg600px outputs the macro `TIC_NUME` which defines this setting to the system information header file "kernel_id.h".
7) Task context register (context)

- Description
Define the register set used by tasks. The settings made here apply to all tasks.

- Definition format
Symbol

- Definition range
Select one from item of “Setting” in Table 20-2.

Table 20-2  system.context

<table>
<thead>
<tr>
<th>Setting</th>
<th>CPU</th>
<th>FPU</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSW, PC, R0 - R7, R14, R15</td>
<td>R8 - R13</td>
<td>FPSW</td>
</tr>
<tr>
<td>NO</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td>FPSW</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>ACC</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td>FPSW,ACC</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>MIN</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td>MIN,FPSW</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
<td>Guaranteed</td>
</tr>
<tr>
<td>MIN,ACC</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td>MON,FPSW,ACC</td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
<td>Guaranteed</td>
</tr>
</tbody>
</table>

a. When compiler option “-isa=rxv2” is specified, the “Accumulator” means ACC0 register and ACC1 register. In the case of others, the “Accumulator” means ACC0 register (in RXv2 architecture) or ACC register (in RXv1 architecture).

Note Compiler option “-isa” is supported by the compiler CC-RX V2.01 or later.

- When omitting
The set value in the default system configuration file (factory setting: NO) applied.

- Note
Be sure to refer to “20.5 Note Concerning system.context”.
20.5 Note Concerning system.context

This section explains note concerning system.context.

20.5.1 Note concerning FPU and DSP

The setting for system.context differs depending on how FPU and DSP are handled. The recommendation setting of system.context is indicated from now on. If other than recommended setting is specified, the RI600PX performance may be slightly deteriorated, compared to the recommended settings case.

1) When using MCU that incorporates FPU and DSP (accumulator)
   Corresponding MCUs: RX600 series, etc.

2) When using MCU that does not incorporate FPU, but incorporates DSP (accumulator)
   Corresponding MCUs: RX200 series, etc.

3) When using MCU that incorporates FPU, but does not incorporate DSP (accumulator)
   Corresponding MCUs: MCUs that correspond to this doesn't exist at the time of making of this manual.

4) When using MCU that incorporate neither FPU nor DSP (accumulator)
   Corresponding MCUs: MCUs that correspond to this doesn't exist at the time of making of this manual.

Note The compiler outputs floating-point arithmetic instructions only when the -fpu option is specified. If the -chkfpu option is specified in the assembler, the floating-point arithmetic instructions written in a program are detected as warning.

In no case does the compiler output the DSP function instructions. If the -chkdsp option is specified in the assembler, the DSP function instructions written in a program are detected as warning.

1) When using MCU that incorporates FPU and DSP (accumulator)

Table 20-3 When using MCU that incorporates FPU and DSP (accumulator)

<table>
<thead>
<tr>
<th>Floating point arithmetic instructions</th>
<th>DSP function instructions</th>
<th>Recommendation setting of system.context</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>“FPSW” and “ACC” included settings essential</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
<td>“FPSW” included setting essential and “ACC” excluded setting recommended</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>“ACC” included setting essential and “FPSW” excluded setting recommended</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
<td>“FPSW” and “ACC” excluded settings recommended</td>
</tr>
</tbody>
</table>
2) When using MCU that does not incorporate FPU, but incorporates DSP (accumulator)

Table 20-4 When using MCU that does not incorporate FPU, but incorporates DSP (accumulator)

<table>
<thead>
<tr>
<th>Usage condition of instruction in tasks</th>
<th>DSP function instructions</th>
<th>Recommendation setting of system.context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point arithmetic instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
<td>Since the MCU does not incorporate FPU, floating-point arithmetic instructions cannot be used.</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>“FPSW” excluded and “ACC” included settings essential</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td>“FPSW” excluded setting essential and “ACC” excluded settings recommended</td>
</tr>
</tbody>
</table>

3) When using MCU that incorporates FPU, but does not incorporate DSP (accumulator)

Table 20-5 When using MCU that incorporates FPU, but does not incorporate DSP (accumulator)

<table>
<thead>
<tr>
<th>Usage condition of instruction in tasks</th>
<th>DSP function instructions</th>
<th>Recommendation setting of system.context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point arithmetic instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
<td>Since the MCU does not incorporate DSP, DSP function instructions cannot be used.</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td>“FPSW” included and “ACC” excluded settings essential</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>Since the MCU does not incorporate DSP, DSP function instructions cannot be used.</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td>“ACC” excluded setting essential and “FPSW” excluded settings recommended</td>
</tr>
</tbody>
</table>
4) When using MCU that incorporate neither FPU nor DSP (accumulator)

Table 20-6 When using MCU that incorporate neither FPU nor DSP (accumulator)

<table>
<thead>
<tr>
<th>Usage condition of instruction in tasks</th>
<th>Recommendation setting of system.context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point arithmetic instructions</td>
<td>DSP function instructions</td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

20.5.2 Relationship with the compiler options “fint_register”, “base” and “pid”

In system.context, by selecting one of choices “MIN,” “MIN, ACC”, “MIN, FPSW,” or “MIN, ACC, FPSW,” it is possible to configure the registers so that R8- R13 registers will not be saved as task context. This results in an increased processing speed.

Note, however, that such a setting of system.context is permitted in only the case where all of R8 - R13 registers are specified to be used by the compiler options “-fint_register”, “-base” and “-pid”.

If, in any other case, the above setting is made for system.context, the kernel will not operate normally.

- Good example:
  1) -fint_register=4 -base=rom=R8 -base=ram=R9
  2) -fint_register=3 -base=rom=R8 -base=ram=R9 -base=0x80000=R10

- Bad example:
  3) No “-fint_register”, “-base” and “-pid” options
  4) -fint_register=4
  5) -base=rom=R8 -base=ram=R9
  6) -fint_register=3 -base=rom=R8 -base=ram=R9
20.6 Base Clock Interrupt Information (clock)

Here, information on the base clock interrupt is defined. The cfg600px outputs the file “ri_cmt.h” where the base clock timer initialization function (__RI_init_cmt()) is described.

Only one “clock” can be defined.

Format

Parentheses < > show the user input part.

```
clock {
    timer       = <1. Selection of timer channel for base clock (timer)>;
    template    = <2. Template file (template)>;
    timer_clock = <3. CMT frequency (timer_clock)>;
    IPL         = <4. Base clock interrupt priority level (IPL)>;
};
```

1) Selection of timer channel for base clock (timer)

- Description
  Define the timer channel for the base clock.

- Definition format
  Symbol

- Definition range
  Select one from Table 20-7.

Table 20-7  clock.timer

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT0</td>
<td>Use CMT channel 0 assigned to relocatable vector 28.</td>
</tr>
<tr>
<td>CMT1</td>
<td>Use CMT channel 1 assigned to relocatable vector 29.</td>
</tr>
<tr>
<td>CMT2</td>
<td>Use CMT channel 2 assigned to relocatable vector 30.</td>
</tr>
<tr>
<td>CMT3</td>
<td>Use CMT channel 3 assigned to relocatable vector 31.</td>
</tr>
<tr>
<td>OTHER</td>
<td>Use a timer other than the above. In this case, the user needs to create a timer initialize routine.</td>
</tr>
<tr>
<td>NOTIMER</td>
<td>Do not use the base clock interrupt.</td>
</tr>
</tbody>
</table>

Note 1  The CMT (Compare Match Timer) is the timer that is mounted on RX MCU typically.

Note 2  Do not select “CMT2” and “CMT3” when CMT channel 2 and channel 3 are not mounted with RX MCU to use, and when relocatable vector assigned to CMT channel 2 and channel 3 is different from Table 20-7 with RX MCU to use.

For example, RX111 does not support CMT channel 2 and channel 3. And in RX64M, relocatable vector assigned to CMT channel 2 and channel 3 is not 30 and 31.

- When omitting
  The set value in the default system configuration file (factory setting: “CMT0”) applied.
2 ) Template file (*template*)

   - **Description**
     Specify template file where hardware information and initialization function of CMT is described. This definition is ignored when either “NOTIMER” or “OTHER” is specified for timer. The template files are provided by the RI600PX. The template files may be added in the future version. Refer to the release notes for MCUs supported by each template file.

     Either CMT1, CMT2 or CMT3 might be unsupported according to template file. When the unsupported CMT channel is specified for timer, the cfg600px does not detect error but the error is detected at compilation of the file which includes “ri_cmt.h”.

   - **Definition format**
     Symbol

   - **Definition range**
     -

   - **When omitting**
     The set value in the default system configuration file (factory setting: “rx630.tpl”) applied.

3 ) CMT frequency (*timer_clock*)

   - **Description**
     Define frequency of the clock supplied to CMT. Please specify the frequency of PCLK (peripheral clock).

   - **Definition format**
     Frequency

   - **Definition range**
     -

   - **When omitting**
     The set value in the default system configuration file (factory setting: “25MHz”) applied.

4 ) Base clock interrupt priority level (*IPL*)

   - **Description**
     Define the interrupt priority level of the base clock interrupt.

   - **Definition format**
     Numeric value

   - **Definition range**
     From 1 to Kernel interrupt mask level (system_IPL) in System Information (system)

   - **When omitting**
     The set value in the default system configuration file (factory setting: 4) applied.

   - **VTIM_LVL**
     The cfg600px outputs the macro VTIM_LVL which defines this setting to the system information header file “kernel_id.h”.
20.7 Maximum ID (maxdefine)

The definition item maxdefine is provided for the definition of the maximum ID for each object. And this definition is required to use service calls to create an object dynamically. The macros in which the maximum ID of each object are defined is output to the system information header file "kernel_id.h". (Refer to "18.2 Constant macros")

Format

Parentheses < > show the user input part.

```
maxdefine {
    max_task    = <1. Maximum task ID (max_task)>
    max_sem     = <2. Maximum semaphore ID (max_sem)>
    max_flag    = <3. Maximum eventflag ID (max_flag)>
    max_dq      = <4. Maximum data queue ID (max_dq)>
    max_mb      = <5. Maximum mailbox ID (max_mb)>
    max_mtx     = <6. Maximum mutex ID (max_mtx)>
    max_mbq     = <7. Maximum message buffer ID (max_mbq)>
    max_mp      = <8. Maximum fixed-sized memory pool ID (max_mp)>
    max_mpi     = <9. Maximum variable-sized memory pool ID (max_mpi)>
    max_cy      = <10. Maximum cyclic handler ID (max_cy)>
    max_al      = <11. Maximum alarm handler ID (max_al)>
    max_da      = <12. Maximum domain ID (max_da)>
};
```

1) Maximum task ID (max_task)

- Description
  The cre_tsk, acre_tsk, del_tsk, exd_tsk and def_tsk can be used by this definition. Ranges of task ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among max_task, the ID number defined in "task[]" and the number of "task[]" definitions
- Definition format
  Numeric value
- Definition range
  From 1 to 255
- When omitting
  The cre_tsk, acre_tsk, del_tsk and def_tsk returns E_NOSPT error. And exd_tsk causes system down. Ranges of task ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among the ID number defined in "task[]" and the number of "task[]" definitions
- VTMAX_TSK
  The cfg600px outputs the macro VTMAX_TSK which defines the maximum value to the system information header file "kernel_id.h".

2) Maximum semaphore ID (max_sem)

- Description
  The cre_sem, acre_sem and del_sem can be used by this definition. Ranges of semaphore ID that can be used are as follows.
  - Minimum value : 1
- Maximum value: The largest one among \texttt{max\_sem}, the ID number defined in "semaphore[]" and the number of "semaphore[]" definitions

- Definition format
  Numeric value

- Definition range
  From 1 to 255

- When omitting
  The \texttt{cre\_sem}, \texttt{acre\_sem} and \texttt{del\_sem} returns E\_NOSPT error.
  Ranges of semaphore ID that can be used are as follows.
    - Minimum value: 1
    - Maximum value: The largest one among the ID number defined in "semaphore[]" and the number of "semaphore[]" definitions

- \texttt{VTMAX\_SEM}
  The \texttt{cfg600px} outputs the macro \texttt{VTMAX\_SEM} which defines the maximum value to the system information header file "kernel\_id.h".

3) Maximum eventflag ID (\texttt{max\_flag})

- Description
  The \texttt{cre\_flg}, \texttt{acre\_flg} and \texttt{del\_flg} can be used by this definition.
  Ranges of eventflag ID that can be used are as follows.
    - Minimum value: 1
    - Maximum value: The largest one among \texttt{max\_flag}, the ID number defined in "flag[]" and the number of "flag[]" definitions

- Definition format
  Numeric value

- Definition range
  From 1 to 255

- When omitting
  The \texttt{cre\_flg}, \texttt{acre\_flg} and \texttt{del\_flg} returns E\_NOSPT error.
  Ranges of eventflag ID that can be used are as follows.
    - Minimum value: 1
    - Maximum value: The largest one among the ID number defined in "flag[]" and the number of "flag[]" definitions

- \texttt{VTMAX\_FLG}
  The \texttt{cfg600px} outputs the macro \texttt{VTMAX\_FLG} which defines the maximum value to the system information header file "kernel\_id.h".

4) Maximum data queue ID (\texttt{max\_dtq})

- Description
  The \texttt{cre\_dtq}, \texttt{acre\_dtq} and \texttt{del\_dtq} can be used by this definition.
  Ranges of data queue ID that can be used are as follows.
    - Minimum value: 1
    - Maximum value: The largest one among \texttt{max\_dtq}, the ID number defined in "data\_queue[]" and the number of "data\_queue[]" definitions

- Definition format
  Numeric value

- Definition range
  From 1 to 255
- When omitting
  The cre_dtq, acre_dtq and del_dtq returns E_NOSPT error.
  Ranges of data queue ID that can be used are as follows.
    - Minimum value : 1
    - Maximum value : The largest one among the ID number defined in "data_queue[]" and the number of
      "data_queue[]" definitions

VTMAX_DTQ
  The cfg600px outputs the macro VTMAX_DTQ which defines the maximum value to the system information
  header file "kernel_id.h".

5 ) Maximum mailbox ID (max_mbx)

- Description
  The cre_mbx, acre_mbx and del_mbx can be used by this definition.
  Ranges of mailbox ID that can be used are as follows.
    - Minimum value : 1
    - Maximum value : The largest one among max_mbx, the ID number defined in "mailbox[]" and the number of
      "mailbox[]" definitions

- Definition format
  Numeric value

- Definition range
  From 1 to 255

- When omitting
  The cre_mbx, acre_mbx and del_mbx returns E_NOSPT error.
  Ranges of mailbox ID that can be used are as follows.
    - Minimum value : 1
    - Maximum value : The largest one among max_mbx, the ID number defined in "mailbox[]" and the number of
      "mailbox[]" definitions

VTMAX_MBX
  The cfg600px outputs the macro VTMAX_MBX which defines the maximum value to the system information
  header file "kernel_id.h".

6 ) Maximum mutex ID (max_mtx)

- Description
  The cre_mtx, acre_mtx and del_mtx can be used by this definition.
  Ranges of mutex ID that can be used are as follows.
    - Minimum value : 1
    - Maximum value : The largest one among max_mtx, the ID number defined in "mutex[]" and the number of
      "mutex[]" definitions

- Definition format
  Numeric value

- Definition range
  From 1 to 255

- When omitting
  The cre_mtx, acre_mtx and del_mtx returns E_NOSPT error.
  Ranges of mutex ID that can be used are as follows.
    - Minimum value : 1
    - Maximum value : The largest one among max_mtx, the ID number defined in "mutex[]" and the number of
      "mutex[]" definitions

VTMAX_MTX
  The cfg600px outputs the macro VTMAX_MTX which defines the maximum value to the system information
7) Maximum message buffer ID (*max_mbf*)

- **Description**
The `cre_mbf`, `acre_mbf` and `del_mbf` can be used by this definition.
Ranges of message buffer ID that can be used are as follows.
- Minimum value : 1
- Maximum value : The largest one among `max_mbf`, the ID number defined in *message_buffer[]* and the number of *message_buffer[]* definitions

- **Definition format**
Numeric value

- **Definition range**
From 1 to 255

- **When omitting**
The `cre_mbf`, `acre_mbf` and `del_mbf` returns E_NOSPT error.
Ranges of message buffer ID that can be used are as follows.
- Minimum value : 1
- Maximum value : The largest one among the ID number defined in *message_buffer[]* and the number of *message_buffer[]* definitions

- **VTMAX_MBF**
The cfg600px outputs the macro `VTMAX_MBF` which defines the maximum value to the system information

8) Maximum fixed-sized memory pool ID (*max_mpf*)

- **Description**
The `cre_mpf`, `acre_mpf` and `del_mpf` can be used by this definition.
Ranges of fixed-sized memory pool ID that can be used are as follows.
- Minimum value : 1
- Maximum value : The largest one among `max_mpf`, the ID number defined in *memorypool[]* and the number of *memorypool[]* definitions

- **Definition format**
Numeric value

- **Definition range**
From 1 to 255

- **When omitting**
The `cre_mpf`, `acre_mpf` and `del_mpf` returns E_NOSPT error.
Ranges of fixed-sized memory pool ID that can be used are as follows.
- Minimum value : 1
- Maximum value : The largest one among the ID number defined in *memorypool[]* and the number of *memorypool[]* definitions

- **VTMAX_MPF**
The cfg600px outputs the macro `VTMAX_MPF` which defines the maximum value to the system information

9) Maximum variable-sized memory pool ID (*max_mpl*)

- **Description**
The `cre_mpl`, `acre_mpl` and `del_mpl` can be used by this definition.
Ranges of variable-sized memory pool ID that can be used are as follows.
- Minimum value : 1
- Maximum value : The largest one among `max_mpl`, the ID number defined in *variable_memorypool[]* and the number of *variable_memorypool[]* definitions

- **Definition format**
Numeric value
- Definition range
  From 1 to 255
- When omitting
  The cre_mpl, acre_mpl and del_mpl returns E_NOSPT error.
  Ranges of variable-sized memory pool ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among the ID number defined in “variable_memorypool[]” and the number of “variable_memorypool[]” definitions
- VTMAX_MPL
  The cfg600px outputs the macro VTMAX_MPL which defines the maximum value to the system information

10 ) Maximum cyclic handler ID (max_cyh)
- Description
  The cre_cyc, acre_cyc and del_cyc can be used by this definition.
  Ranges of cyclic handler ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among max_cyh, the ID number defined in “cyclic_hand[]” and the number of “cyclic_hand[]” definitions
- Definition format
  Numeric value
- Definition range
  From 1 to 255
- When omitting
  The cre_cyc, acre_cyc and del_cyc returns E_NOSPT error.
  Ranges of cyclic handler ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among the ID number defined in “cyclic_hand[]” and the number of “cyclic_hand[]” definitions
- VTMAX_CYH
  The cfg600px outputs the macro VTMAX_CYH which defines the maximum value to the system information

11 ) Maximum alarm handler ID (max_alh)
- Description
  The cre_alm, acre_alm and del_alm can be used by this definition.
  Ranges of alarm handler ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among max_alh, the ID number defined in “alarm_hand[]” and the number of “alarm_hand[]” definitions
- Definition format
  Numeric value
- Definition range
  From 1 to 255
- When omitting
  The cre_alm, acre_alm and del_alm returns E_NOSPT error.
  Ranges of alarm handler ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among the ID number defined in “alarm_hand[]” and the number of “alarm_hand[]” definitions
- **VTMAX_ALH**
  The cfg600px outputs the macro VTMAX_ALH which defines the maximum value to the system information

12 ) Maximum domain ID (max_domain)

- **Description**
  Ranges of domain ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among max_domain, the ID number defined in “domain[]”

- **Definition format**
  Numeric value

- **Definition range**
  From 1 to 15

- **When omitting**
  Ranges of mutex ID that can be used are as follows.
  - Minimum value : 1
  - Maximum value : The largest one among the ID number defined in “domain[]”

- **VTMAX_DOMAIN**
  The cfg600px outputs the macro VTMAX_DOMAIN which defines the maximum value to the system information
### 20.8 Domain Definition (domain[])

Here, each domain is defined. The domain that is not defined by this static API is handled as “trust = NO”.

#### Format

Parentheses < > show the user input part.

```plaintext
domain[ <1. ID number> ] { 
  trust          = <2. Trusted domain (trust)>;
};
```

1) ID number

- **Description**
  Define the domain ID number.

- **Definition format**
  Numeric value

- **Definition range**
  From 1 to 15

- **When omitting**
  Cannot be omitted.

2) Trusted domain (trust)

- **Description**
  Define whether the domain is Trusted Domain.

- **Definition format**
  Symbol

- **Definition range**
  Select either of the following:
  - **YES:** The domain is trusted domain.
  - **OFF:** The domain is not trusted domain.

- **When omitting**
  The set value in the default system configuration file (factory setting: "NO") applied.
20.9 Memory Object Definition (memory_object[])  

Here, each memory object is defined. 
Please be sure to refer to “3.11 Design of Memory Map”. 

Format 
Parentheses < > show the user input part. 

```
memory_object[] { 
    start_address = <1. Start address of memory object (start_address)>; 
    end_address   = <2. Termination address of memory object (end_address)>; 
    acptn1        = <3. Access permission pattern (acptn1, acptn2, acptn2)>; 
    acptn2        = <3. Access permission pattern (acptn1, acptn2, acptn2)>; 
    acptn3        = <3. Access permission pattern (acptn1, acptn2, acptn2)>; 
};
```

1) Start address of memory object (start_address) 

- Description 
  Define start address of the memory object by numeric value or section name. 
  When section name is specified, since the section should be started from 16-bytes boundary address, specify 
  “aligned_section” linker option at linking. 
  When numeric value is specified, the value must be multiple of 16. 

- Definition format 
  Symbol or numeric value 

- Definition range 
  Numeric value : From 0 to 0xFFFFFFF0, and multiple of 16 

- When omitting 
  Cannot be omitted. 

2) Termination address of memory object (end_address) 

- Description 
  Define termination address of the memory object by numeric value or section name. 
  When section name is specified, the address in which termination address of this section is rounded up to 
  “multiple of 16 + 15” is treated with the termination address of the memory object. If the termination address 
  of this section is not “multiple of 16 + 15”, you must not allocate any section in the range from “the termination 
  address of this section + 1” to next “multiple of 16 + 15” address. The above-mentioned conditions are fulfilled by 
  specifying “aligned_section” linker option as the section which follows this section at linking. 

- When numeric value is specified, the value must be multiple of 16 + 15. 

- Definition format 
  Symbol or numeric value 

- Definition range 
  Numeric value : From 0x0000000F to 0xFFFFFFFF, and multiple of 16 + 15 

- When omitting 
  Cannot be omitted. 

3) Access permission pattern (acptn1, acptn2, acptn2) 

- Description 
  Define access permission pattern for operand-read access (acptn1), operand-write access (acptn2) and execution 
  access (acptn3) by symbol or numeric value. 
  Only the “TACP_SHARED” (permit access to all the domains) can be specified as symbol. 
  Use numeric value to specify permission for each domain according to Figure 20-1.
When either of the bits corresponding to domains exceeding the maximum domain ID is set, the cfg600px does not detect an error and it becomes a **SYSTEM DOWN** at starting the RI600PX.

Figure 20-1 Access Permission Pattern

- **Definition format**
  - Symbol or numeric value

- **Definition range**
  - Symbol: TACP_SHARED: Permits access to all the domains.
  - Numeric value: From 0 to 0x7FFF

- **When omitting**
  - The set value in the default system configuration file (factory setting: “TACP_SHARED”) applied.
20.10 Task Information (task[])

Here, each task is defined.

Format

Parentheses < > show the user input part.

```
task[ <1. ID number> ] {
  name          = <2. ID name (name)>;
  entry_address = <3. Task entry address (entry_address)>;
  stack_size    = <4. User stack size (stack_size)>;
  stack_section = <5. Section name assigned to the stack area (stack_section)>;
  priority      = <6. Task initial priority (priority)>;
  initial_start = <7. TA_ACT attribute (initial_start)>;
  exinf         = <8. Extended information (exinf)>;
  texrtn        = <9. Task exception handling routine entry address (texrtn)>;
  domain_num    = <10. Belonging domain ID (domain_num)>;
}
```

1) ID number
- Description
  Define the task ID number.
- Definition format
  Numeric value
- Definition range
  From 1 to 255
- When omitting
  The cfg600px assigns the ID number automatically.

2) ID name (name)
- Description
  Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.
  ```
  #define <ID name> <ID number>
  ```
- Definition format
  Symbol
- Definition range
  -
- When omitting
  Cannot be omitted.

3) Task entry address (entry_address)
- Description
  Define the starting function of the task.
- Definition format
  Symbol
- Definition range
  -
- When omitting
  Cannot be omitted.
4) User stack size (stack_size)
- Description
  Define the user stack size.
- Definition format
  Numeric value
- Definition range
  More than the following values, and multiple of 16.

Table 20-8 Lower Bound Value of User Stack Size

<table>
<thead>
<tr>
<th>Setting of system.context</th>
<th>Compiler option &quot;-isa&quot;</th>
<th>Lower bound value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>-</td>
<td>68</td>
</tr>
<tr>
<td>FPSW</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>ACC</td>
<td>&quot;-isa=rxv2&quot;</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>&quot;-isa=rxv1&quot; or not specify &quot;-isa&quot;</td>
<td>76</td>
</tr>
<tr>
<td>FPSW,ACC</td>
<td>&quot;-isa=rxv2&quot;</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>&quot;-isa=rxv1&quot; or not specify &quot;-isa&quot;</td>
<td>80</td>
</tr>
<tr>
<td>MIN</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>MIN, FPSW</td>
<td>-</td>
<td>48</td>
</tr>
<tr>
<td>MIN, ACC</td>
<td>&quot;-isa=rxv2&quot;</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>&quot;-isa=rxv1&quot; or not specify &quot;-isa&quot;</td>
<td>52</td>
</tr>
<tr>
<td>MON, FPSW, ACC</td>
<td>&quot;-isa=rxv2&quot;</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>&quot;-isa=rxv1&quot; or not specify &quot;-isa&quot;</td>
<td>56</td>
</tr>
</tbody>
</table>

Note
- Compiler option "-isa" is supported by the compiler CC-RX V2.01 or later.
- When omitting
  The set value in the default system configuration file (factory setting: 256) applied.

5) Section name assigned to the stack area (stack_section)
- Description
  Define the section name to be assigned to the user stack area.
  The cfg600px generates the user stack area with the size specified by stack_size to the section specified by stack_section. The section attribute is "DATA", and the alignment number is 4.
  When linking, be sure to locate this section in the RAM area. Note, this section must not be located to address 0. Since this section should be started from 16-bytes boundary address, specify "aligned_section" linker option at linking.
- Definition format
  Symbol
- Definition range
  -
- When omitting
  The set value in the default system configuration file (factory setting: "SURI_STACK") applied.
6) Task initial priority (priority)
   - Description
     Define the task initial priority.
   - Definition format
     Numeric value
   - Definition range
     From 1 to Maximum task priority (priority) in System Information (system)
   - When omitting
     The set value in the default system configuration file (factory setting: 1) applied.

7) TA_ACT attribute (initial_start)
   - Description
     Define the initial state of the task.
   - Definition format
     Symbol
   - Definition range
     Select either of the following:
     - ON: Specify the TA_ACT attribute. (The initial state is READY state.)
     - OFF: Not Specify the TA_ACT attribute. (The initial state is DORMANGT state.)
   - When omitting
     The set value in the default system configuration file (factory setting: “OFF”) applied.

8) Extended information (exinf)
   - Description
     Define the extended information of the task.
   - Definition format
     Numeric value
   - Definition range
     From 0 to 0xFFFFFFFF
   - When omitting
     The set value in the default system configuration file (factory setting: 0) applied.
   - Note
     When the task is activated by the TA_ACT attribute, act_tsk or iact_tsk, the extended information is passed to the task.

9) Task exception handling routine entry address (texrtn)
   - Description
     Define the starting function of the task exception handling routine. To not define task exception handling routine, do not define texrtn.
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     The task exception handling routine is not defined.

10) Belonging domain ID (domain_num)
    - Description
      Define the ID number of the domain by which the task belongs.
- Definition format
  Numeric value

- Definition range
  From 1 to 15

- When omitting
  The set value in the default system configuration file (factory setting: 1) applied.
20.11 Semaphore Information (semaphore[])

Here, each semaphore is defined.

Format

Parentheses < > show the user input part.

```c
semaphore[ <1. ID number> ] {
    name = <2. ID name (name)>;
    max_count = <3. Maximum resource count (max_count)>;
    initial_count = <4. Initial resource count (initial_count)>;
    wait_queue = <5. Wait queue attribute (wait_queue)>;
};
```

1) ID number

- Description
  Define the semaphore ID number.

- Definition format
  Numeric value

- Definition range
  From 1 to 255

- When omitting
  The cfg600px assigns the ID number automatically.

2) ID name (name)

- Description
  Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.

```c
#define <ID name> <ID number>
```

- Definition format
  Symbol

- Definition range
  -

- When omitting
  Cannot be omitted.

3) Maximum resource count (max_count)

- Description
  Define the maximum resource count

- Definition format
  Numeric value

- Definition range
  From 1 to 65535

- When omitting
  The set value in the default system configuration file (factory setting: 1) applied.
4) Initial resource count (initial_count)
   - Description
     Define the initial resource count.
   - Definition format
     Numeric value
   - Definition range
     From 0 to max_count
   - When omitting
     The set value in the default system configuration file (factory setting: 1) applied.

5) Wait queue attribute (wait_queue)
   - Description
     Define the wait queue attribute.
   - Definition format
     Symbol
   - Definition range
     Select either of the following:
     TA_TFIFO: FIFO order
     TA_TPRI: Task current priority order
     Among tasks with the same current priority, they are queued in FIFO order.
   - When omitting
     The set value in the default system configuration file (factory setting: “TA_TFIFO”) applied.
20.12 Eventflag Information (flag[])  

Here, each semaphore is defined.

Format  
Parentheses < > show the user input part.

flag[ <1. ID number> ] {  
    name = <2. ID name (name)>;
    initial_pattern = <3. Initial bit pattern (initial_pattern)>;
    wait_multi = <4. Multiple wait permission attribute (wait_multi)>;
    clear_attribute = <5. Clear attribute (clear_attribute)>;
    wait_queue = <6. Wait queue attribute (wait_queue)>;
};

1) ID number  
   - Description  
     Define the eventflag ID number.
   - Definition format  
     Numeric value
   - Definition range  
     From 1 to 255
   - When omitting  
     The cfg600px assigns the ID number automatically.

2) ID name (name)  
   - Description  
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.

     #define <ID name> <ID number>
   - Definition format  
     Symbol
   - Definition range  
     -
   - When omitting  
     Cannot be omitted.

3) Initial bit pattern (initial_pattern)  
   - Description  
     Define the initial bit pattern
   - Definition format  
     Numeric value
   - Definition range  
     From 0 to 0xFFFFFFFF
   - When omitting  
     The set value in the default system configuration file (factory setting: 0) applied.
4) Multiple wait permission attribute (wait_multi)
   - Description
     Define the attribute regarding whether multiple tasks are permitted to wait for the eventflag.
   - Definition format
     Symbol
   - Definition range
     Select either of the following:
     - TA_WSGL: Not permit multiple tasks to wait for the eventflag.
     - TA_WMUL: Permit multiple tasks to wait for the eventflag.
   - When omitting
     The set value in the default system configuration file (factory setting: “TA_WSGL”) applied.

5) Clear attribute (clear_attribute)
   - Description
     Define the clear attribute (TA_CLR).
   - Definition format
     Symbol
   - Definition range
     Select either of the following:
     - NO: Not specify the TA_CLR attribute.
     - YES: Specify the TA_CLR attribute.
   - When omitting
     The set value in the default system configuration file (factory setting: “NO”) applied.

6) Wait queue attribute (wait_queue)
   - Description
     Define the wait queue attribute.
   - Definition format
     Symbol
   - Definition range
     Select either of the following: However, when the TA_CLR attribute is not specified, the wait queue is managed in the FIFO order even if TA_TPRI is specified for wait_queue. This behavior falls outside μITRON4.0 specification.
     - TA_TFIFO: FIFO order
     - TA_TPRI: Task current priority order
       Among tasks with the same current priority, they are queued in FIFO order.
   - When omitting
     The set value in the default system configuration file (factory setting: “TA_TFIFO”) applied.
20.13 Data Queue Information (dataqueue[])

Here, each data queue is defined.

Format

Parentheses < > show the user input part.

```
dataqueue[ <1. ID number> ] { 
    name          = <2. ID name (name)>;
    buffer_size   = <3. Data count (buffer_size)>;
    wait_queue    = <4. Wait queue attribute (wait_queue)>;
};
```

1) ID number
   - Description
     Define the data queue ID number.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 255
   - When omitting
     The cfg600px assigns the ID number automatically.

2) ID name (name)
   - Description
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.

     ```
     #define <ID name> <ID number>
     ```
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

3) Data count (buffer_size)
   - Description
     Define the number of data that the data queue can be stored.
   - Definition format
     Numeric value
   - Definition range
     From 0 to 65535
   - When omitting
     The set value in the default system configuration file (factory setting: 0) applied.
4) Wait queue attribute (`wait_queue`)

- **Description**
  Define the wait queue attribute for sending.
  Note, task wait queue for receiving is managed in FIFO order.

- **Definition format**
  Symbol

- **Definition range**
  Select either of the following:

  - **TA_TFIFO**: FIFO order
  - **TA_TPRI**: Task current priority order

  Among tasks with the same current priority, they are queued in FIFO order.

- **When omitting**
  The set value in the default system configuration file (factory setting: “TA_TFIFO”) applied.
20.14 Mailbox Information (mailbox[])

Here, each mailbox is defined.

Format

Parentheses < > show the user input part.

```c
mailbox[ <1. ID number> ] {
  name = <2. ID name (name)>;
  wait_queue = <3. Wait queue attribute (wait_queue)>;
  message_queue = <4. Message queue attribute (message_queue)>;
  max_pri = <5. Maximum message priority (max_pri)>;
}
```

1) ID number
   - Description
     Define the mailbox ID number.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 255
   - When omitting
     The cfg600px assigns the ID number automatically.

2) ID name (name)
   - Description
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.
     ```
     #define <ID name> <ID number>
     ```
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

3) Wait queue attribute (wait_queue)
   - Description
     Define the wait queue attribute.
   - Definition format
     Symbol
   - Definition range
     Select either of the following:
     - TA_TFIFO: FIFO order
     - TA_TPRI: Task current priority order
       Among tasks with the same current priority, they are queued in FIFO order.
   - When omitting
     The set value in the default system configuration file (factory setting: “TA_TFIFO”) applied.

4) Message queue attribute (message_queue)
- Description
  Define the message queue attribute.

- Definition format
  Symbol

- Definition range
  Select either of the following:
  TA_MFIFO: The order of the message transmission request.
  TA_MPRI: Message priority order

- When omitting
  The set value in the default system configuration file (factory setting: “TA_MFIFO”) applied.

5) Maximum message priority (max_pri)

- Description
  When TA_MPRI is specified for message_queue, the message priority from 1 to max_pri can be used.
  When TA_MFIFO is specified for message_queue, this item is only disregarded.

- Definition format
  Numeric value

- Definition range
  From 1 to Maximum message priority (message_pri) in System Information (system)

- When omitting
  The set value in the default system configuration file (factory setting: 1) applied.
20.15 Mutex Information (mutex[])  

Here, each mutex is defined.

**Format**  
Parentheses < > show the user input part.

```plaintext
mutex[<1. ID number>] {  
    name = <2. ID name (name)>;  
    ceilpri = <3. Ceiling priority (ceilpri)>;  
};
```

1) ID number  
- Description  
  Define the mutex ID number.
- Definition format  
  Numeric value
- Definition range  
  From 1 to 255
- When omitting  
  The cfg600px assigns the ID number automatically.

2) ID name (name)  
- Description  
  Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.

```plaintext
#define <ID name> <ID number>
```
- Definition format  
  Symbol
- Definition range  
  -
- When omitting  
  Cannot be omitted.

3) Ceiling priority (ceilpri)  
- Description  
  The RI600PX adopts *Simplified priority ceiling protocol*. The ceiling priority should be defined in ceilpri.
- Definition format  
  Numeric value
- Definition range  
  From 1 to *Maximum task priority (priority)* in *System Information (system)*
- When omitting  
  The set value in the default system configuration file (factory setting: 1) applied.
20.16 Message Buffer Information (message_buffer[])  
Here, each message buffer is defined. 

**Format**  
Parentheses < > show the user input part.  

```
message_buffer[<1. ID number>] { 
    name        = <2. ID name (name)>;
    mbf_size    = <3. Buffer size (mbf_size)>;
    mbf_section = <4. Section name assigned to the message buffer area (mbf_section)>;
    max_msgsz   = <5. Maximum message size (max_msgsz)>;
};
```

1) ID number  
- **Description**  
  Define the message buffer ID number.  
- **Definition format**  
  Numeric value  
- **Definition range**  
  From 1 to 255  
- **When omitting**  
  The cfg600px assigns the ID number automatically.  

2) ID name (name)  
- **Description**  
  Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.  
  ```
  #define <ID name> <ID number>
  ```  
- **Definition format**  
  Symbol  
- **Definition range**  
  -  
- **When omitting**  
  Cannot be omitted.  

3) Buffer size (mbf_size)  
- **Description**  
  Define the size of the message buffer in bytes.  
- **Definition format**  
  Numeric value  
- **Definition range**  
  0, or multiple of 4 in the range from 8 to 65532  
- **When omitting**  
  The set value in the default system configuration file (factory setting: 0) applied.
4) Section name assigned to the message buffer area (mbf_section)

- Description
  Define the section name to be assigned to the message buffer area.
  When mbf_size > 0, the cfg600px generates the message buffer area with the size specified by buffer_size to the section specified by mbf_section. The section attribute is “DATA”, and the alignment number is 4.
  When linking, be sure to locate this section in the RAM area. Note, this section must not be located to address 0.

- Definition format
  Symbol

- Definition range
  -

- When omitting
  The set value in the default system configuration file (factory setting: “BURI_HEAP”) applied.

5) Maximum message size (max_msgsz)

- Description
  Define the maximum message size of the message buffer in bytes.
  When mbf_size > 0, max_msgsz must be less than or equal to “mbf_size - 4”.

- Definition format
  Numeric value

- Definition range
  From 1 to 65528

- When omitting
  The set value in the default system configuration file (factory setting: 4) applied.
20.17 Fixed-sized Memory Pool Information (memorypool[])  
Here, each fixed-sized memory pool is defined.

Format  
Parentheses < > show the user input part.

```
memorypool[ <1. ID number> ] {
    name       = <2. ID name (name)>;
    siz_block  = <3. The size of the fixed-sized memory block (siz_block)>;
    num_block  = <4. The number of the fixed-sized memory block (num_block)>;
    section    = <5. Section name assigned to the memory pool area (section)>
    wait_queue = <6. Wait queue attribute (wait_queue)>;
};
```

1) ID number  
- Description  
Define the fixed-sized memory pool ID number.

- Definition format  
Numeric value

- Definition range  
From 1 to 255

- When omitting  
The cfg600px assigns the ID number automatically.

2) ID name (name)  
- Description  
Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.

```
#define   <ID name>   <ID number>
```

- Definition format  
Symbol

- Definition range  
-

- When omitting  
Cannot be omitted.

3) The size of the fixed-sized memory block (siz_block)  
- Description  
Define the size of the fixed-sized memory block in bytes.

- Definition format  
Numeric value

- Definition range  
From 1 to 65535

- When omitting  
The set value in the default system configuration file (factory setting: 256) applied.
4) The number of the fixed-sized memory block \((num\_block)\)

- **Description**
  Define the number of the fixed-sized memory block.

- **Definition format**
  Numeric value

- **Definition range**
  From 1 to 65535

- **When omitting**
  The set value in the default system configuration file (factory setting: 1) applied.

5) Section name assigned to the memory pool area \((section)\)

- **Description**
  Define the section name to be assigned to the fixed-sized memory pool area.
  The cfg600px generates the fixed-sized memory pool area with the size calculated by \("siz\_block \times num\_block"\) to
  the section specified by \(section\). The section attribute is \("DATA"\), and the alignment number is 4.
  When linking, be sure to locate this section in the RAM area. Note, this section must not be located to address 0.

- **Definition format**
  Symbol

- **Definition range**
  -

- **When omitting**
  The set value in the default system configuration file (factory setting: "BRI_HEAP") applied.

6) Wait queue attribute \((wait\_queue)\)

- **Description**
  Define the wait queue attribute.

- **Definition format**
  Symbol

- **Definition range**
  Select either of the following:
  
  - **TA_TFIFO**: FIFO order
  - **TA_TPRI**: Task current priority order
    Among tasks with the same current priority, they are queued in FIFO order.

- **When omitting**
  The set value in the default system configuration file (factory setting: “TA_TFIFO”) applied.
20.18 Variable-sized Memory Pool Information (variable_memorypool[])

Here, each variable-sized memory pool is defined.

**Format**

Parentheses `< >` show the user input part.

```plaintext
variable_memorypool[ <1. ID number> ] {  
  name       = <2. ID name (name)> ;  
  heap_size  = <3. The size of the variable-sized memory pool (heap_size)> ;  
  num_block  = <4. Upper limit of the variable-sized memory block (max_memsiz)< > ;  
  section    = <5. Section name assigned to the memory pool area (mpl_section)> ;  
};
```

1) ID number
   - Description
     Define the variable-sized memory pool ID number.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 255
   - When omitting
     The cfg600px assigns the ID number automatically.

2) ID name (name)
   - Description
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.
     ```plaintext
     #define   <ID name>   <ID number>
     ```
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

3) The size of the variable-sized memory pool (heap_size)
   - Description
     Define the size of the variable-sized memory pool area in bytes.
   - Definition format
     Numeric value
   - Definition range
     From 24 to 0x10000000
   - When omitting
     The set value in the default system configuration file (factory setting: 1024) applied.
4 ) Upper limit of the variable-sized memory block (max_memsize)
   - Description
     Define the upper limit of an acquirable memory block size in bytes.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 0xBFFFFF4
   - When omitting
     The set value in the default system configuration file (factory setting: 36) applied.
   - Note
     Refer to “9.3.1 Size of Variable-sized memory block” for the size of the variable-sized memory blocks.

5 ) Section name assigned to the memory pool area (mpl_section)
   - Description
     Define the section name to be assigned to the variable-sized memory pool area.
     The cfg600px generates the variable-sized memory pool area with the size specified by heap_size to the section specified by mpl_section. The section attribute is “DATA”, and the alignment number is 4.
     When linking, be sure to locate this section in the RAM area. Note, this section must not be located to address 0.
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     The set value in the default system configuration file (factory setting: “BRI_HEAP”) applied.
20.19 Cyclic Handler Information (cyclic_hand[])

Here, each cyclic handler is defined.

Format

Parentheses < > show the user input part.

```c
    cyclic_hand[ <1. ID number> ] {
        name = <2. ID name (name)>;
        entry_address = <3. Cyclic handler entry address (entry_address)>;
        interval_counter = <4. Activation cycle (interval_counter)>;
        start = <5. Initial state (start)>;
        phs_counter = <6. Activation phase (phs_counter)>;
        phsatr = <7. TA_PHS attribute (phsatr)>;
        exinf = <8. Extended information (exinf)>;
    };
```

1) ID number
   - Description
     Define the cyclic handler ID number.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 255
   - When omitting
     The cfg600px assigns the ID number automatically.

2) ID name (name)
   - Description
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.
     
     ```
     #define <ID name> <ID number>
     ```
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

3) Cyclic handler entry address (entry_address)
   - Description
     Define the starting function of the cyclic handler.
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

4) Activation cycle (interval_counter)
- **Description**
  Define the activation cycle in millisecond.

- **Definition format**
  Numeric value

- **Definition range**
  From 1 to \( (0x7FFFFFFF - \text{system.tic_nume}) / \text{system.tic_deno} \)

- **When omitting**
  The set value in the default system configuration file (factory setting: 1) applied.

5 ) **Initial state** (start)

- **Description**
  Define the initial state of the cyclic handler.

- **Definition format**
  Symbol

- **Definition range**
  Select either of the following:
  
  OFF: Non operational stat (The TA_STA attribute is not specified.)
  
  ON: Operational state (The TA_STA attribute is specified.)

- **When omitting**
  The set value in the default system configuration file (factory setting: “OFF”) applied.

6 ) **Activation phase** (phs_counter)

- **Description**
  Define the activation phase in millisecond

- **Definition format**
  Numeric value

- **Definition range**
  From 0 to \( \text{interval_counter} \)

- **When omitting**
  The set value in the default system configuration file (factory setting: 0) applied.

7 ) **TA_PHS attribute** (phsatr)

- **Description**
  Define the attribute concerning the activation phase.

- **Definition format**
  Symbol

- **Definition range**
  Select either of the following:
  
  OFF: Not preserve the activation phase. (The TA_PHS attribute is not specified.)
  
  ON: Preserve the activation phase. (The TA_PHS attribute is specified.)

- **When omitting**
  The set value in the default system configuration file (factory setting: “OFF”) applied.

8 ) **Extended information** (exinf)

- **Description**
  Define the extended information of the cyclic handler.

- **Definition format**
  Numeric value

- **Definition range**
  From 0 to 0xFFFFFFFF
- When omitting
  The set value in the default system configuration file (factory setting: 0) applied.

- Note
  The extended information is passed to the cyclic handler.
20.20 Alarm Handler Information (alarm_handl[])

Here, each alarm handler is defined.

**Format**

Parentheses < > show the user input part.

```c
alarm_handl[ <1. ID number> ] {
    name        = <2. ID name (name)>;
    entry_address = <3. Alarm handler entry address (entry_address)>;
    exinf       = <4. Extended information (exinf)>;
};
```

1 ) ID number
   - Description
     Define the alarm handler ID number.
   - Definition format
     Numeric value
   - Definition range
     From 1 to 255
   - When omitting
     The cfg600px assigns the ID number automatically.

2 ) ID name (name)
   - Description
     Define the ID name. The specified ID name is output to the system information header file (kernel_id.h) in the form of the following.
     ```
     #define <ID name>   <ID number>
     ```
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.

3 ) Alarm handler entry address (entry_address)
   - Description
     Define the starting function of the alarm handler.
   - Definition format
     Symbol
   - Definition range
     -
   - When omitting
     Cannot be omitted.
4) Extended information (exinf)

- Description
  Define the extended information of the alarm handler.

- Definition format
  Numeric value

- Definition range
  From 0 to 0xFFFFFFFF

- When omitting
  The set value in the default system configuration file (factory setting: 0) applied.

- Note
  The extended information is passed to the alarm handler.
20.21 Relocatable Vector Information (interrupt_vector[])  
Here, each interrupt handler for relocatable vector of the RX MCU is defined.  
If any interrupt occurs whose vector number is not defined here, the system goes down.  
Note, the cfg600px does not generate code to initialize the interrupt control registers, the causes of interrupts, etc. for the interrupts defined here. These initialization need to be implemented in the application.  

Note Since the vector number from 1 to 8 are reserved by the RI600PX, do not define these vectors. And do not define the vectors which are reserved by the MCU specification.

Format  
Parentheses < > show the user input part.

interrupt_vector[ <1. Vector number> ] {  
    entry_address = <2. Interrupt handler entry address (entry_address)>;  
    os_int = <3. Kernel interrupt specification (os_int)>;  
    pragma_switch = <4. Switch passed to pragma directive (pragma_switch)>;  
};

1 ) Vector number  
   - Description  
     Define the vector number.  
   - Definition format  
     Numeric value  
   - Definition range  
     From 0 to 255  
   - When omitting  
     Cannot be omitted.

2 ) Interrupt handler entry address (entry_address)  
   - Description  
     Define the starting function of the interrupt handler.  
   - Definition format  
     Symbol  
   - Definition range  
     -  
   - When omitting  
     Cannot be omitted.

3 ) Kernel interrupt specification (os_int)  
   - Description  
     Interrupts whose interrupt priority level is lower than or equal to the Kernel interrupt mask level (system_IPL) must be defined as the kernel interrupt, and the other interrupts must be defined as the non-kernel interrupt.
     Note, when the Kernel interrupt mask level (system_IPL) is 15, all interrupts for relocatable vector must be defined as the kernel interrupt.  
   - Definition format  
     Symbol  
   - Definition range  
     Select either of the following:  
       YES: Kernel interrupt  
       NO: Non-kernel interrupt
Switch passed to pragma directive (pragma_switch)

- Description
The cfg600px outputs "#pragma interrupt" directive to handle the function specified by entry_address as an interrupt function to the system information header file kernel_id.h.
The switches passed to this pragma directive should be specified for pragma_switch.

- Definition format
Symbol

- Definition range
The following can be specified. To specify multiple choices, separate each with a comma. However, "ACC" and "NOACC" cannot be specified at the same time.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>The &quot;enable&quot; switch that permits a multiple interrupt is passed.</td>
</tr>
<tr>
<td>F</td>
<td>The &quot;fint&quot; switch that specifies a fast interrupt is passed. Note, a fast interrupt must be handled as non-kernel interrupt (os_int = NO).</td>
</tr>
<tr>
<td>S</td>
<td>The &quot;save&quot; switch that limits the number of registers used in the interrupt handler is passed.</td>
</tr>
<tr>
<td>ACC</td>
<td>The &quot;acc&quot; switch that guarantees the ACC register in the interrupt handler is passed.</td>
</tr>
<tr>
<td>NOACC</td>
<td>The &quot;no_acc&quot; switch that does not guarantee the ACC register in the interrupt handler is passed</td>
</tr>
</tbody>
</table>

- When omitting
No switches are passed.

Note 1 Refer to Table 20-9 for the guarantee of the ACC register.

Table 20-9 Guarantee of the ACC Register

<table>
<thead>
<tr>
<th>Setting of pragma_switch</th>
<th>&quot;-save_acc&quot; compiler option</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not specified</td>
</tr>
<tr>
<td>Neither &quot;ACC&quot; nor &quot;NOACC&quot; is not specified.</td>
<td>Neither &quot;acc&quot; nor &quot;no_acc&quot; switch is not passed. The ACC register is not guaranteed.</td>
</tr>
<tr>
<td>&quot;ACC&quot; is specified.</td>
<td>The &quot;acc&quot; switch is passed. The ACC register is guaranteed.</td>
</tr>
<tr>
<td>&quot;NOACC&quot; is specified.</td>
<td>The &quot;no_acc&quot; switch is passed. The ACC register is not guaranteed.</td>
</tr>
</tbody>
</table>

Note 2 When either "CMT0", "CMT1", "CMT2" or "CMT3" is defined as Selection of timer channel for base clock (timer), it is treated that "interrupt_vector[]" is implicitly defined by the following specification.

- Vector number
  - CMT0 : 28
  - CMT1 : 29
  - CMT2 : 30
  - CMT3 : 31

- entry_address : The entry address of the base clock interrupt processing routine in the RI600PX
- os_int : YES
- `pragma_switch` : E,ACC
20.22 Fixed Vector/Exception Vector Information (interrupt_fvector[])

Here, fixed vector table of the RXv1 architecture (address from 0xFFFFFF80 to 0xFFFFFFFF) / exception vector table of RXv2 architecture is defined. Not only interrupt handler address but also the endian select register, etc., are included in fixed vector table/exception vector table.

All interrupt in fixed vector/exception vector is non-kernel interrupt.

In the RI600PX, the vector number is allocated according to the vector address as shown in Table 20-10. The Table 20-10 also shows the setting of the vector to which the definition is omitted.

Note, the content of fixed vector table/exception vector table is different in each MCU. For details, refer to the hardware manual of the MCU used.

Note, the cfg600px does not generate code to initialize the interrupt control registers, the causes of interrupts, etc. for the interrupts defined here. These initialization need to be implemented in the application.

Table 20-10 Fixed Vector Table/Exception Vector table

<table>
<thead>
<tr>
<th>Vector address a</th>
<th>Vector number</th>
<th>Example of factor (different in each MCU)</th>
<th>When omitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFF80</td>
<td>0</td>
<td>Endian select register</td>
<td>The following are set according to “-endian” compiler option. - “-endian=little” 0xFFFFFFFF - “-endian=big” 0xFFFFFFF8</td>
</tr>
<tr>
<td>0xFFFFFF84</td>
<td>1</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFF88</td>
<td>2</td>
<td>Option function select register 1</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFF8C</td>
<td>3</td>
<td>Option function select register 0</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFF90</td>
<td>4</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFF94</td>
<td>5</td>
<td>(Reserved area)</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>0xFFFFFF98</td>
<td>6</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFF9C</td>
<td>7</td>
<td>ROM code protection (flash memory)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFA0</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFA4</td>
<td>9</td>
<td>ID code protection on connection of the on-chip debugger (flash memory)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFA8</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFAC</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFB0</td>
<td>12</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFB4</td>
<td>13</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFB8</td>
<td>14</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFBC</td>
<td>15</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>Vector address a</td>
<td>Vector number</td>
<td>Example of factor (different in each MCU)</td>
<td>When omitting</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0xFFFFFFFCC0</td>
<td>16</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFCC4</td>
<td>17</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFCC8</td>
<td>18</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFCC</td>
<td>19</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFDD0</td>
<td>20</td>
<td>Privileged instruction exception</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFDD4</td>
<td>21</td>
<td>Access exception</td>
<td>Access exception handler b</td>
</tr>
<tr>
<td>0xFFFFFFFDD8</td>
<td>22</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFDC</td>
<td>23</td>
<td>Undefined instruction exception</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFEE0</td>
<td>24</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFEE4</td>
<td>25</td>
<td>Floating-point exception</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFEE8</td>
<td>26</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFEC</td>
<td>27</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFF0</td>
<td>28</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFF4</td>
<td>29</td>
<td>(Reserved area)</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFF8</td>
<td>30</td>
<td>Non-maskable interrupt</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFCC</td>
<td>31</td>
<td>Reset</td>
<td>PowerON_Reset_PC()</td>
</tr>
</tbody>
</table>

* a. The vector address in Table 20-10 is the address of fixed vector table in RXv1 architecture. The address of exception vector table in RXv2 architecture is decided by EXTB register. The initial value of EXTB register at the time of reset is same as fixed vector table in RXv1 architecture. Refer to "FIX_INTERRUPT_VECTOR section" in section 2.6.4.
* b. Do not define a handler to the vector-21. If defined, the access exception handler never be initiated.

**Format**

Parentheses < > show the user input part.

```c
interrupt_fvector[ <1. Vector number> ] { 
   entry_address = <2. Interrupt handler entry address (entry_address)>;
   pragma_switch = <3. Switch passed to pragma directive (pragma_switch)>;
};
```

1) Vector number
   - Description
     Define the vector number.
   - Definition format
     Numeric value
   - Definition range
     From 0 to 31
- When omitting
  Cannot be omitted.

2) Interrupt handler entry address (entry_address)

- Description
  Define the starting function of the interrupt handler or the set value to fixed vector/exception vector.

- Definition format
  Symbol or numeric value

- Definition range
  From 0 to 0xFFFFFFFF when a numeric value is specified.

- When omitting
  Cannot be omitted.

3) Switch passed to pragma directive (pragma_switch)

- Description
  The cfg600px outputs "#pragma interrupt" directive to handle the function specified by entry_address as a interrupt function to the system information header file kernel_id.h.
  The switches passed to this pragma directive should be specified for pragma_switch.

- Definition format
  Symbol

- Definition range
  The following can be specified. To specify multiple choices, separate each with a comma. However, ”ACC” and ”NOACC” cannot be specified at the same time.

  S: The “save” switch that limits the number of registers used in the interrupt handler is passed.

  ACC: The “acc” switch that guarantees the ACC register in the interrupt handler is passed.

  NOACC: The “no_acc” switch that does not guarantee the ACC register in the interrupt handler is passed.

- When omitting
  No switches are passed.

- Note
  Refer to Table 20-9 for the guarantee of the ACC register.
20.23 RAM Capacity Estimation

Memory areas used and managed by the RI600PX are broadly classified into four types of sections.

- **BRI_RAM section**: The RI600PX’s management data, data queue area created by the system configuration file, message buffer area created by the system configuration file without specifying section
- **RRI_RAM section**: The RI600PX’s management data (when dynamic creation function is used)
- **BURI_HEAP section**: Fixed-sized memory pool area and variable-sized memory pool area created by the system configuration file without specifying section
- **SURI_STACK section**: User stack area of tasks created by the system configuration file without specifying section
- **SI section**: System stack area

### 20.23.1 BRI_RAM and RRI_RAM section

The RI600PX’s management data is located in the BRI_RAM and RRI_RAM section. The Table 20-11 shows the size calculation method for the BRI_RAM and RRI_RAM section (unit: bytes). In addition, actual size may become larger than the value computed by Table 20-11 for boundary adjustment.

#### Table 20-11 BRI_RAM and RRI_RAM Section Size Calculation Method

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Section</th>
<th>Size Calculation Method (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System control block</td>
<td>BRI_RAM</td>
<td>(28 + 4 \times \text{down} \left( \frac{TMAX_TPRI}{32} + 1 \right) + TMAX_TPRI) + (VMAX_SEM + 2 \times VMAX_DTQ + VMAX_FLG + VMAX_MBX) + (VMAX_MTX + 2 \times VMAX_MBF + VMAX_MPF + VMAX_MPL) + (57 \times VMAX_DOMAIN)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^a)</td>
<td>(4 + VMAX_SEM + 2 \times VMAX_DTQ + VMAX_FLG + VMAX_MBX) + (VMAX_MTX + 2 \times VMAX_MBF + VMAX_MPF + VMAX_MPL)</td>
</tr>
<tr>
<td>Task control block</td>
<td>BRI_RAM</td>
<td>28 (\times VMAX_TSK)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^b)</td>
<td>24 (\times VMAX_TSK)</td>
</tr>
<tr>
<td>Semaphore control block(^c)</td>
<td>BRI_RAM</td>
<td>4 (\times VMAX_SEM + \text{down} \left( \frac{VMAX_SEM}{8} + 1 \right))</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^d)</td>
<td>4 (\times VMAX_SEM)</td>
</tr>
<tr>
<td>Eventflag control block(^e)</td>
<td>BRI_RAM</td>
<td>8 (\times VMAX_FLG + 2 \times \text{down} \left( \frac{VMAX_FLG}{8} + 1 \right))</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^f)</td>
<td>4 (\times VMAX_FLG)</td>
</tr>
<tr>
<td>Data queue control block(^g)</td>
<td>BRI_RAM</td>
<td>6 (\times VMAX_DTQ + \text{down} \left( \frac{VMAX_DTQ}{8} + 1 \right) + DTQ_ALLSIZE)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^h)</td>
<td>8 (\times VMAX_DTQ)</td>
</tr>
<tr>
<td>Mailbox control block(^i)</td>
<td>BRI_RAM</td>
<td>8 (\times VMAX_MBX + 2 \times \text{down} \left( \frac{VMAX_MBX}{8} + 1 \right))</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^j)</td>
<td>VMAX_MBX</td>
</tr>
<tr>
<td>Mutex control block(^k)</td>
<td>BRI_RAM</td>
<td>VMAX_MTX + \text{down} \left( \frac{VMAX_MTX}{8} + 1 \right)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^l)</td>
<td>VMAX_MTX</td>
</tr>
<tr>
<td>Message buffer control block(^m)</td>
<td>BRI_RAM</td>
<td>16 (\times VMAX_MBF + MBF_ALLSIZE)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^n)</td>
<td>8 (\times VMAX_MBF)</td>
</tr>
<tr>
<td>Fixed-sized memory pool control block(^o)</td>
<td>BRI_RAM</td>
<td>8 (\times VMAX_MPF + \text{down} \left( \frac{VMAX_MPF}{8} + 1 \right) + \sum \text{memorypool[i].num_block} / 8 + 1)</td>
</tr>
<tr>
<td></td>
<td>RRI_RAM(^p)</td>
<td>12 (\times VMAX_MPF)</td>
</tr>
</tbody>
</table>
Variable-sized memory pool control block
BRI_RAM $36 \times VTMAX\_MPL$
RRI\_RAM^f $20 \times VTMAX\_MPL$

Cyclic handler control block
BRI\_RAM $8 \times VTMAX\_CYH$
RRI\_RAM^f $20 \times VTMAX\_CYH$

Alarm handler control block
BRI\_RAM $8 \times VTMAX\_ALH$
RRI\_RAM^y $8 \times VTMAX\_ALH$

a. When all Maximum task ID (max_task), Maximum semaphore ID (max_sem), Maximum eventflag ID (max_flag), Maximum data queue ID (max_dtq), Maximum mailbox ID (max_mbx), Maximum mutex ID (max_mtx), Maximum message buffer ID (max_mbf), Maximum fixed-sized memory pool ID (max_mpf) and Maximum variable-sized memory pool ID (max_mpl) are 0 or undefined, the size of this area is 4 bytes.
b. This area is generated only when Maximum task ID (max_task) is defined.
c. This area is not generated when VTMAX\_SEM is 0.
d. This area is generated only when Maximum semaphore ID (max_sem) is defined.
e. This area is not generated when VTMAX\_FLG is 0.
f. This area is generated only when Maximum eventflag ID (max_flag) is defined.
g. This area is not generated when VTMAX\_DTQ is 0.
h. This area is generated only when Maximum data queue ID (max_dtq) is defined.
i. This area is not generated when VTMAX\_MBX is 0.
j. This area is generated only when Maximum mailbox ID (max_mbx) is defined.
k. This area is not generated when VTMAX\_MTX is 0.
l. This area is generated only when Maximum mutex ID (max_mtx) is defined.
m. This area is not generated when VTMAX\_MBF is 0.
n. This area is generated only when Maximum message buffer ID (max_mbf) is defined.
o. This area is not generated when VTMAX\_MPF is 0.
p. This area is generated only when Maximum fixed-sized memory pool ID (max_mpf) is defined.
q. This area is not generated when VTMAX\_MPF is 0.
r. This area is generated only when Maximum variable-sized memory pool ID (max_mpl) is defined.
s. This area is not generated when VTMAX\_CYH is 0.
t. This area is generated only when Maximum cyclic handler ID (max_cyh) is defined.
u. This area is not generated when VTMAX\_ALH is 0.
v. This area is generated only when Maximum alarm handler ID (max_alh) is defined.
Note Each keyword in the size calculation methods has the following meaning.

**TMAX_TPRI:** The TMAX_TPRI represents maximum task priority. The cfg600px outputs the macro TMAX_TPRI which defines the value set as Maximum task priority (priority) in System Information (system) to the system information header file kernel_id.h.

**VTMAX_TSK:** The VTMAX_TSK represents the maximum task ID. The cfg600px outputs the macro of VTMAX_TSK to the system information header file kernel_id.h. For details, refer to “Maximum task ID (max_task)”.

**VTMAX_SEM:** The VTMAX_SEM represents the maximum semaphore ID. The cfg600px outputs the macro of VTMAX_SEM to the system information header file kernel_id.h. For details, refer to “Maximum semaphore ID (max_sem)”. 

**VTMAX_FLG:** The VTMAX_FLG represents the maximum eventflag ID. The cfg600px outputs the macro of VTMAX_FLG to the system information header file kernel_id.h. For details, refer to “Maximum eventflag ID (max_flag)”. 

**VTMAX_DTQ:** The VTMAX_DTQ represents the maximum data queue ID. The cfg600px outputs the macro of VTMAX_DTQ to the system information header file kernel_id.h. For details, refer to “Maximum data queue ID (max_dtq)”. 

**DTQ_ALLSIZE:** Total of size of data queue area created in the system configuration file. Concretely, it is calculated by the following expressions.
\[
\sum \text{dataqueue[]}\text{.buffer\_size} \times 4
\]
Note, DTQ_ALLSIZE is 4 when this calculation result is 0.

**VTMAX_MBX:** The VTMAX_MBX represents the maximum mailbox ID. The cfg600px outputs the macro of VTMAX_MBX to the system information header file kernel_id.h. For details, refer to “Maximum mailbox ID (max_mbx)”. 

**VTMAX_MTX:** The VTMAX_MTX represents the maximum mutex ID. The cfg600px outputs the macro of VTMAX_MTX to the system information header file kernel_id.h. For details, refer to “Maximum mutex ID (max_mtx)”. 

**VTMAX_MBF:** The VTMAX_MBF represents the maximum message buffer ID. The cfg600px outputs the macro of VTMAX_MBF to the system information header file kernel_id.h. For details, refer to “Maximum message buffer ID (max_mbf)”. 

**MBF_ALLSIZE:** Total of size of message buffer area created in the system configuration file without specifying “mbf_section”. Concretely, it is calculated by the following expressions.
\[
\sum \text{message\_buffer[]}\text{.mbf\_size} \times 4
\]

**VTMAX_MPF:** The VTMAX_MPF represents the maximum fixed-sized memory pool ID. The cfg600px outputs the macro of VTMAX_MPF to the system information header file kernel_id.h. For details, refer to “Maximum fixed-sized memory pool ID (max_mpf)”. 

**VTMAX_MPL:** The VTMAX_MPL represents the maximum variable-sized memory pool ID. The cfg600px outputs the macro of VTMAX_MPL to the system information header file kernel_id.h. For details, refer to “Maximum variable-sized memory pool ID (max_mpl)”. 

**VTMAX_CYH:** The VTMAX_CYH represents the maximum cyclic handler ID. The cfg600px outputs the macro of VTMAX_CYH to the system information header file kernel_id.h. For details, refer to “Maximum cyclic handler ID (max_cyh)”. 

**VTMAX_ALH:** The VTMAX_ALH represents the maximum alarm handler ID. The cfg600px outputs the macro of VTMAX_ALH to the system information header file kernel_id.h. For details, refer to “Maximum alarm handler ID (max_alh)”. 

**VTMAX_DOMAIN:** The VTMAX_DOMAIN represents the maximum domain ID. The cfg600px outputs the macro of VTMAX_DOMAIN to the system information header file kernel_id.h. For details, refer to “Maximum domain ID (max_domain)”. 

**DTQ_ALLSIZE:** Total of size of data queue area created in the system configuration file. Concretely, it is calculated by the following expressions.
\[
\sum \text{dataqueue[]}\text{.buffer\_size} \times 4
\]
Note, DTQ_ALLSIZE is 4 when this calculation result is 0.
20.23.2 BURI_HEAP section

The fixed-sized memory pool area and variable-sized memory pool area are located in the BURI_HEAP section. Note, when a fixed-sized memory pool and variable-sized memory pool are defined, the area can be located into the user-specific section.

The size of the BURI_HEAP section is calculated by the total of following. In addition, when user specific data is generated in the BURI_HEAP section, the size should be added.

- Total size of fixed-sized memory pool area
  This is calculated about the definition of Fixed-sized Memory Pool Information (memorypool[]) that omits to specify “section” by the following expressions.

  \[ \sum (\text{memorypool[]} \cdot \text{siz}_\text{block} \times \text{memorypool[]} \cdot \text{num}_\text{block}) \]

- Total size of variable-sized memory pool area
  This is calculated about the definition of Variable-sized Memory Pool Information (variable_memorypool[]) that omits to specify “mpl_section” by the following expressions.

  \[ \sum \text{variable_memorypool[]} \cdot \text{heap}_\text{size} \]

Note For estimation of stack size, refer to “APPENDIX D STACK SIZE ESTIMATION”.

20.23.3 SURI_STACK section

The user stack area is located in the SURI_STACK section. Note, when a task is defined, the user stack area can be located into the user-specific section.

The size of the SURI_STACK section is calculated about the definition of Task Information (task[]) that omits to specify “stack_section” by the following expressions. In addition, when user specific data is generated in the SURI_STACK section, the size should be added.

\[ \sum \text{task[]} \cdot \text{stack}_\text{size} \]

Note For estimation of stack size, refer to “APPENDIX D STACK SIZE ESTIMATION”.

20.23.4 SI section

The system stack area is located in the SI section.

The system stack size is the same as a set value for System stack size (stack_size) in System Information (system).

Note For estimation of stack size, refer to “APPENDIX D STACK SIZE ESTIMATION”.
20.24 Description Examples

The following describes an example for coding the system configuration file.

Note The RI600PX provides sample source files for the system configuration file.

// System Definition
system{
    stack_size = 1024;
    priority = 10;
    system_IPL = 14;
    message_pri = 1;
    tic_deno = 1;
    tic_nume = 1;
    context = FPSW,ACC;
};

// System Clock Definition ----------------------------------------
clock{
    timer = CMT0;
    template = rx630.tpl;
    timer_clock = 25MHz;
    IPL = 13;
};

// Number of object ----------------------------------------
max_task = 10;
max_sem = 1;
max_flag = ;
max_dtq = 1;
max_mbx = ;
max_mtx = ;
max_mbf = ;
max_mpf = ;
max_mpl = 1;
max_cyh = ;
max_alh = ;
max_domain = 3;
}

// Trusted domain
domain[1] {
    trust = YES;
};

// Memory Object Definition : Master domain data
memory_object[1]{
    start_address = BU_MASTERDOM;
    end_address = RU_MASTERDOM_2;
    acptn1 = 0x0001;
    acptn2 = 0x0001;
    acptn3 = 0;
};

// Memory Object Definition : App-domain A data
memory_object[2]{
    start_address = BU_DOM_A;
    end_address = RU_DOM_A_2;
    acptn1 = 0x0002;
    acptn2 = 0x0002;
    acptn3 = 0;
};
// Memory Object Definition : App-domain B data
memory_object[3]{
    start_address = BU_DOM_B;
    end_address = RU_DOM_B_2;
    acptn1 = 0x0004;
    acptn2 = 0x0004;
    acptn3 = 0;
};

// Memory Object Definition : Shared data
memory_object[4]{
    start_address = BURI_HEAP;
    end_address = RU_SH_2;
    acptn1 = TACP_SHARED;
    acptn2 = TACP_SHARED;
    acptn3 = 0;
};

// Memory Object Definition : Master domain code and const
memory_object[5]{
    start_address = PU_MASTERDOM;
    end_address = DU_MASTERDOM_2;
    acptn1 = 0x0001;
    acptn2 = 0;
    acptn3 = 0x0001;
};

// Memory Object Definition : App-domain A code and const
memory_object[6]{
    start_address = PU_DOM_A;
    end_address = DU_DOM_A_2;
    acptn1 = 0x0002;
    acptn2 = 0;
    acptn3 = 0x0002;
};

// Memory Object Definition : App-domain B code and const
memory_object[7]{
    start_address = PU_DOM_B;
    end_address = DU_DOM_B_2;
    acptn1 = 0x0004;
    acptn2 = 0;
    acptn3 = 0x0004;
};

// Memory Object Definition : Shared code and const
memory_object[8]{
    start_address = PU_SH;
    end_address = DU_SH_2;
    acptn1 = TACP_SHARED;
    acptn2 = 0;
    acptn3 = TACP_SHARED;
};
```
// Task Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_MASTERDOMTASK</td>
</tr>
<tr>
<td>entry_address</td>
<td>MasterDom_Task();</td>
</tr>
<tr>
<td>initial_start</td>
<td>ON;</td>
</tr>
<tr>
<td>stack_size</td>
<td>256;</td>
</tr>
<tr>
<td>priority</td>
<td>1;</td>
</tr>
</tbody>
</table>

// Stack section = SURI_STACK;

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>exinf</td>
<td>1;</td>
</tr>
<tr>
<td>texrtn</td>
<td>;</td>
</tr>
<tr>
<td>domain_num</td>
<td>1;</td>
</tr>
</tbody>
</table>

// Semaphore Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_SEM1;</td>
</tr>
<tr>
<td>wait_queue</td>
<td>TA_TFIFO;</td>
</tr>
<tr>
<td>max_count</td>
<td>1;</td>
</tr>
<tr>
<td>initial_count</td>
<td>1;</td>
</tr>
</tbody>
</table>

// Eventflag Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_FLG1;</td>
</tr>
<tr>
<td>initial_pattern</td>
<td>0;</td>
</tr>
<tr>
<td>wait_queue</td>
<td>TA_TFIFO;</td>
</tr>
<tr>
<td>wait_multi</td>
<td>TA_WSGL;</td>
</tr>
<tr>
<td>clear_attribute</td>
<td>NO;</td>
</tr>
</tbody>
</table>

// Data Queue Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_DTQ1;</td>
</tr>
<tr>
<td>buffer_size</td>
<td>4;</td>
</tr>
<tr>
<td>wait_queue</td>
<td>TA_TFIFO;</td>
</tr>
</tbody>
</table>

// Mailbox Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_MBX1;</td>
</tr>
<tr>
<td>wait_queue</td>
<td>TA_TFIFO;</td>
</tr>
<tr>
<td>message_queue</td>
<td>TA_MFIFO;</td>
</tr>
<tr>
<td>max_pri</td>
<td>1;</td>
</tr>
</tbody>
</table>

// Mutex definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_MTX1;</td>
</tr>
<tr>
<td>ceilpri</td>
<td>1;</td>
</tr>
</tbody>
</table>

// Message Buffer Definition ----------------------------------------

##

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ID_MBF1;</td>
</tr>
<tr>
<td>mbf_size</td>
<td>128;</td>
</tr>
<tr>
<td>mbf_section</td>
<td>BRI_RAM;</td>
</tr>
<tr>
<td>max_msgsz</td>
<td>16;</td>
</tr>
<tr>
<td>wait_queue</td>
<td>TA_TFIFO;</td>
</tr>
</tbody>
</table>
```

// Fixed-sized Memory Pool Definition ----------------------------------------
memorypool[]{
  name            = ID_MPF1;
  section         = BURI_HEAP;
  num_block       = 1;
  siz_block       = 0x100;
  wait_queue      = TA_TFIFO;
};

// Variable-sized Memory Pool Definition ----------------------------------------
variable_memorypool[]{
  name            = ID_MPL1;
  mpl_section     = BURI_HEAP;
  heap_size       = 1024;
  max_memsize     = 36;
};

// Cyclic Handler Definition ----------------------------------------
cyclic_hand[] {
  name            = ID_CYC1;
  entry_address   = cyh1();
  interval_counter = 10;
  start           = ON;
  phsatr          = OFF;
  phs_counter     = 10;
  exinf           = 1;
};

// Alarm Handler Definition ----------------------------------------
alarm_hand[] {
  name            = ID_ALM1;
  entry_address   = alh1();
  exinf           = 1;
};

// Relocatable Vector Definition ----------------------------------------
interrupt_vector[64]{
  os_int          = YES;
  entry_address   = inh64();
  pragma_switch   = E;
};

// Fixed Vector Definition ----------------------------------------
interrupt_fvector[0]{// MDES register (address : 0xFFFFFFF80)
  entry_address   = AUTO_ENDIAN;
};

interrupt_fvector[1]{// Reserved (address : 0xFFFFFFFF84)
  entry_address   = 0xFFFFFFFF;
};

interrupt_fvector[2]{// OFS1 register (address : 0xFFFFFFFF88)
  entry_address   = 0xFFFFFFFF;
};

interrupt_fvector[3]{// OFS0 register (address : 0xFFFFFFFF8C)
  entry_address   = 0xFFFFFFFF;
};

interrupt_fvector[4]{// Reserved (address : 0xFFFFFFFF90)
  entry_address   = 0xFFFFFFFF;
};
// interrupt_fvector[5]{// Reserved (address : 0xFFFFFFFF96)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[6]{// Reserved (address : 0xFFFFFFFF98)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[7]{// ROM code protect (address : 0xFFFFFFFF9C)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[8]{// ID code protect (address : 0xFFFFFFFFA0)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[9]{// ID code protect (address : 0xFFFFFFFFA4)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[10]{// ID code protect (address : 0xFFFFFFFFA8)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[11]{// ID code protect (address : 0xFFFFFFFFAC)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[12]{// Reserved (address : 0xFFFFFFFFA0)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[13]{// Reserved (address : 0xFFFFFFFFA4)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[14]{// Reserved (address : 0xFFFFFFFFA8)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[15]{// Reserved (address : 0xFFFFFFFFAC)
  entry_address   = 0xFFFFFFFF;
// }

// interrupt_fvector[30]{// NMI (address : 0xFFFFFFFF8)
  entry_address   = NMI_handler();
  pragma_switch   = ;
// }

// interrupt_fvector[31]{// Reset (address : 0xFFFFFFFFC)
  entry_address   = PowerON_Reset_PC();
// }
CHAPTER 21  CONFIGURATOR cfg600px

This chapter explains configurator cfg600px.

21.1 Outline

To build systems (load module) that use functions provided by the RI600PX, the information storing data to be provided for
the RI600PX is required.
Since information files are basically enumerations of data, it is possible to describe them with various editors.
Information files, however, do not excel in descriptiveness and readability; therefore substantial time and effort are
required when they are described.
To solve this problem, the RI600PX provides a utility tool (configurator “cfg600px”) that converts a system configuration file
which excels in descriptiveness and readability into information files.
The cfg600px reads the system configuration file as a input file, and then outputs information files.
The information files output from the cfg600px are explained below.

- System information header file (kernel_id.h)
  An information file that contains the correspondence between object names (task names, semaphore names, or the
  like) described in the system configuration file and IDs.

- Service call definition file (kernel_sysint.h)
  The declaration for issuing service calls by using INT instruction is described in this file. This file is included by ker-
  nel.h.

- ROM definition file (kernel_rom.h), RAM definition file (kernel_ram.h)
  These files contain the RI600PX management data. These files must be included only by the boot processing file. For
details, refer to “17.2 Boot Processing File (User-Own Coding Module)”.

- System definition file (ri600.inc)
  The system definition file is included by the table file (ritable.src) which is generated by the mktitbl.

- Vector table template file (vector.tpl)
  The vector table template file is input to the mkritblpx.

- CMT timer definition file (ri_cmt.h)
  When either of CMT0, CMT1, CMT or CMT3 is specified for Selection of timer channel for base clock (timer) for in
Base Clock Interrupt Information (clock), the Template file (template) is retrieved from the folder indicated by the envi-
ronment variable “LIB600”, and the retrieved file is output after it is renamed to “ri_cmt.h”. The CMT timer definition
file is used for the base clock timer initialization routine. For details, refer to “10.9 Base Clock Timer Initialization
Routine (_RI_init_cmt_knl( ))”.


21.2 Start cfg600px

21.2.1 Start cfg600px from command line

It is necessary to set the environment variable “LIB600” to “<ri_root>\lib600” beforehand. The following is how to activate the cfg600px from the command line. Note that, in the examples below, “C>” indicates the command prompt, “Δ” indicates pressing of the space key, and “<Enter>” indicates pressing of the enter key. The options enclosed in ‘[]’ can be omitted.

```
C> cfg600px.exe Δ [-U] Δ [-v] Δ [-V] Δ file <Enter>
```

The output files are generated to the current folder.

The details of each option are explained below:

- **-U**
  When an undefined interrupt occurs, the system down is caused. When -U option is specified, the vector number will be transferred to the system down routine (refer to "CHAPTER 15 SYSTEM DOWN"). This is useful for debugging. However, the kernel code size increases by about 1.5 kB.

- **-v**
  Show a description of the command option and details of its version.

- **-V**
  Show the creation status of files generated by the cfg600px.

- **file**
  Specifies the system configuration file name to be input. If the filename extension is omitted, the extension “.cfg” is assumed.

Note

*<ri_root> indicates the installation folder of RI600PX.
The default folder is “C:\Program Files\Renesas Electronics\CubeSuite+\RI600PX”.

21.2.2 Start cfg600px from CubeSuite+

This is started when CubeSuite+ performs a build, in accordance with the setting on the Property panel, on the [System Configuration File Related Information] tab.
CHAPTER 22  TABLE GENARATION UTILITY mkritblpx

This chapter explains the table generation utility mkritblpx.

22.1 Outline

The utility mkritblpx is a command line tool that after collecting service call information used in the application, generates service call tables and interrupt vector tables.

When compiling applications, the service call information files (.mrc) that contains the service call information to be used are generated. The mkribl reads the service call information files, and generates the service call table to be linked only the service calls used in the system.

Furthermore, the mkritblpx generates an interrupt vector table based on the vector table template files generated by the cfg600px and the service call information files.

Figure 22-1 Outline of mkritblpx

The short dashed arrow represents "include", and solid arrow represents "input/output file".
22.2 Start mkritblpx

22.2.1 Start mkritblpx from command line

It is necessary to set the environment variable “LIB600” to “<ri_root>\lib600” beforehand. The following is how to activate the mkritblpx from the command line. Note that, in the examples below, “C>” indicates the command prompt, “Δ” indicates pressing of the space key, and “<Enter>” indicates pressing of the enter key. The options enclosed in “[ ]” can be omitted.

```
C> mkritblpx.exe Δ [path] <Enter>
```

The output files are generated to the current folder.

The details of each option are explained below:

- **path**
  Specifies the service call information file or the path to the folder where the service call information files are retrieved. Note, when a folder path is specified, the sub folder is not retrieved. The mkritblpx makes the current folder a retrieval path regardless of this specification.

Note 
<ri_root> indicates the installation folder of RI600PX. The default folder is “C:\Program Files\Renesas Electronics\CubeSuite+\RI600PX”.

22.2.2 Start mkritblpx from CubeSuite+

This is started when CubeSuite+ performs a build, in accordance with the setting on the Property panel, on the [System Configuration File Related Information] tab.

22.3 Notes

Refer to “2.6.1 Service call information files and “-ri600_preinit_mrc” compiler option”. 
This appendix explains the window/panels that are used when the activation option for the configurator cfg600px and the table generation utility mkritblpx is specified from the integrated development environment CubeSuite+.

A.1 Description

The following shows the list of window/panels.

<table>
<thead>
<tr>
<th>Window/Panel Name</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main window</td>
<td>This is the first window to be open when CubeSuite+ is launched.</td>
</tr>
<tr>
<td>Project Tree panel</td>
<td>This panel is used to display the project components in tree view.</td>
</tr>
<tr>
<td>Property panel</td>
<td>This panel is used to display the detailed information on the Realtime OS node, system configuration file, or the like that is selected on the Project Tree panel and change the settings of the information.</td>
</tr>
</tbody>
</table>
Main window

Outline

This is the first window to be open when CubeSuite+ is launched. This window is used to control the user program execution and open panels for the build process.

This window can be opened as follows:

- Select Windows [start] -> [All programs] -> [Renesas Electronics CubeSuite+] -> [CubeSuite+]

Display image
Explanation of each area

1) Menu bar
Displays the menus relate to realtime OS.
Contents of each menu can be customized in the User Setting dialog box.

- [View]

```
<table>
<thead>
<tr>
<th>Realtime OS</th>
<th>The [View] menu shows the cascading menu to start the tools of realtime OS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Information</td>
<td>Opens the Realtime OS Resource Information panel. Note that this menu is disabled when the debug tool is not connected.</td>
</tr>
</tbody>
</table>
```

2) Toolbar
Displays the buttons relate to realtime OS.
Buttons on the toolbar can be customized in the User Setting dialog box. You can also create a new toolbar in the same dialog box.

- Realtime OS toolbar

```
| ![Resource Information](image) | Opens the Realtime OS Resource Information panel. Note that this button is disabled when the debug tool is not connected. |
```

3) Panel display area
The following panels are displayed in this area.

- Project Tree panel
- Property panel
- Output panel

See the each panel section for details of the contents of the display.

Project Tree panel

Outline

This panel is used to display the project components such as Realtime OS node, system configuration file, etc. in tree view.

This panel can be opened as follows:

- From the [View] menu, select [Project Tree].

Display image

![Project Tree panel image]
Explanation of each area

1) Project tree area

Project components are displayed in tree view with the following given node.

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI600PX (Realtime OS) (referred to as “Realtime OS node”)</td>
<td>Realtime OS to be used.</td>
</tr>
<tr>
<td>xxx.cfg</td>
<td>System configuration file.</td>
</tr>
</tbody>
</table>

Realtime OS generated files (referred to as “Realtime OS generated files node”) |

- System information header file (kernel_id.h)
- Service call definition file (kernel_sysint.h)
- ROM definition file (kernel_rom.h)
- RAM definition file (kernel_ram.h)
- System definition file (ri600.inc)
- vector table template file (vector.tpl)
- CMT timer definition file (ri_cmt.h)

This node and files displayed under this node cannot be deleted directly.
This node and files displayed under this node will no longer appear if you remove the system configuration file from the project.

Context menu

1) When the Realtime OS node or Realtime OS generated files node is selected

| Property | Displays the selected node's property on the Property panel. |

2) When the system configuration file or an information file is selected

<table>
<thead>
<tr>
<th>Assemble</th>
<th>Assembles the selected assembler source file. Note that this menu is only displayed when a system information table file or an entry file is selected. Note that this menu is disabled when the build tool is in operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Opens the selected file with the application corresponds to the file extension. Note that this menu is disabled when multiple files are selected.</td>
</tr>
<tr>
<td>Open with Internal Editor...</td>
<td>Opens the selected file with the Editor panel. Note that this menu is disabled when multiple files are selected.</td>
</tr>
<tr>
<td>Open with Selected Application...</td>
<td>Opens the Open with Program dialog box to open the selected file with the designated application. Note that this menu is disabled when multiple files are selected.</td>
</tr>
<tr>
<td>Open Folder with Explorer</td>
<td>Opens the folder that contains the selected file with Explorer.</td>
</tr>
<tr>
<td>Add</td>
<td>Shows the cascading menu to add files and category nodes to the project.</td>
</tr>
<tr>
<td>Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Add File...</td>
<td>Opens the Add Existing File dialog box to add the selected file to the project.</td>
</tr>
<tr>
<td>Add New File...</td>
<td>Opens the Add File dialog box to create a file with the selected file type and add to the project.</td>
</tr>
<tr>
<td>Add New Category</td>
<td>Adds a new category node at the same level as the selected file. You can rename the category. This menu is disabled while the build tool is running, and if categories are nested 20 levels.</td>
</tr>
<tr>
<td>Remove from Project</td>
<td>Removes the selected file from the project. The file itself is not deleted from the file system. Note that this menu is disabled when the build tool is in operation.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected file to the clipboard. When the file name is in editing, the characters of the selection are copied to the clipboard.</td>
</tr>
<tr>
<td>Paste</td>
<td>This menu is always disabled.</td>
</tr>
<tr>
<td>Rename</td>
<td>You can rename the selected file. The actual file is also renamed.</td>
</tr>
<tr>
<td>Property</td>
<td>Displays the selected file's property on the Property panel.</td>
</tr>
</tbody>
</table>
Property panel

Outline
This panel is used to display the detailed information on the Realtime OS node, system configuration file, or the like that is selected on the Project Tree panel by every category and change the settings of the information.

This panel can be opened as follows:

- On the Project Tree panel, select the Realtime OS node, system configuration file, or the like, and then select the [View] menu -> [Property] or the [Property] from the context menu.

Note When either one of the Realtime OS node, system configuration file, or the like on the Project Tree panel while the Property panel is opened, the detailed information of the selected node is displayed.

Display image

Explanation of each area

1) Selected node area
   Display the name of the selected node on the Project Tree panel.
   When multiple nodes are selected, this area is blank.

2) Detailed information display/change area
   In this area, the detailed information on the Realtime OS node, system configuration file, or the like that is selected on the Project Tree panel is displayed by every category in the list. And the settings of the information can be changed directly.
   Mark [ ] indicates that all the items in the category are expanded. Mark [ ] indicates that all the items are collapsed.
   You can expand/collapse the items by clicking these marks or double clicking the category name.
   See the section on each tab for the details of the display/setting in the category and its contents.

3) Property description area
   Display the brief description of the categories and their contents selected in the detailed information display/change area.
4) Tab selection area

Categories for the display of the detailed information are changed by selecting a tab. In this panel, the following tabs are contained (see the section on each tab for the details of the display/setting on the tab).

- When the Realtime OS node is selected on the Project Tree panel
  - [RI600PX] tab

- When the system configuration file is selected on the Project Tree panel
  - [System Configuration File Related Information] tab
  - [File Information] tab

- When the Realtime OS generated files node is selected on the Project Tree panel
  - [Category Information] tab

- When the system information table file or entry file is selected on the Project Tree panel
  - [Build Settings] tab
  - [Individual Assemble Options] tab
  - [File Information] tab

- When the system information header file is selected on the Project Tree panel
  - [File Information] tab


Note2 When multiple components are selected on the Project Tree panel, only the tab that is common to all the components is displayed. If the value of the property is modified, that is taken effect to the selected components all of which are common to all.

[Edit] menu (only available for the Project Tree panel)

<table>
<thead>
<tr>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Cancels the previous edit operation of the value of the property.</td>
</tr>
<tr>
<td>Cut</td>
<td>While editing the value of the property, cuts the selected characters and copies them to the clip board.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected characters of the property to the clip board.</td>
</tr>
<tr>
<td>Paste</td>
<td>While editing the value of the property, inserts the contents of the clip board.</td>
</tr>
<tr>
<td>Delete</td>
<td>While editing the value of the property, deletes the selected character string.</td>
</tr>
<tr>
<td>Select All</td>
<td>While editing the value of the property, selects all the characters of the selected property.</td>
</tr>
</tbody>
</table>

Context menu

<table>
<thead>
<tr>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Cancels the previous edit operation of the value of the property.</td>
</tr>
<tr>
<td>Cut</td>
<td>While editing the value of the property, cuts the selected characters and copies them to the clip board.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected characters of the property to the clip board.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Paste</td>
<td>While editing the value of the property, inserts the contents of the clip board.</td>
</tr>
<tr>
<td>Delete</td>
<td>While editing the value of the property, deletes the selected character string.</td>
</tr>
<tr>
<td>Select All</td>
<td>While editing the value of the property, selects all the characters of the selected property.</td>
</tr>
<tr>
<td>Reset to Default</td>
<td>Restores the configuration of the selected item to the default configuration of the project. For the [Individual Assemble Options] tab, restores to the configuration of the general option.</td>
</tr>
<tr>
<td>Reset All to Default</td>
<td>Restores all the configuration of the current tab to the default configuration of the project. For the [Individual Assemble Options] tab, restores to the configuration of the general option.</td>
</tr>
</tbody>
</table>
[ RI600PX ] tab

Outline
This tab shows the detailed information on RI600PX to be used categorized by the following.

- Version Information

Display image

<table>
<thead>
<tr>
<th>Property</th>
<th>RI600PX Property</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version Information</th>
<th>Show variable properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel version</td>
<td>V1.01.00</td>
</tr>
<tr>
<td>Install folder</td>
<td>C:\Program Files\Renesas Electronics\CubeSuite+\RI600PX</td>
</tr>
<tr>
<td>Endian</td>
<td>Little endian</td>
</tr>
</tbody>
</table>

**Kernel version**
This is the version of the RI600PX to be used in this project.

Explanation of each area

1) [Version Information]
The detailed information on the version of the RI600PX are displayed.

<table>
<thead>
<tr>
<th>Kernel version</th>
<th>Display the version of RI600PX to be used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>The version of the installed RI600PX</td>
</tr>
<tr>
<td>How to change</td>
<td>Changes not allowed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Install folder</th>
<th>Display the folder in which RI600PX to be used is installed with the absolute path.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>The folder in which RI600PX to be used is installed</td>
</tr>
<tr>
<td>How to change</td>
<td>Changes not allowed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Endian</th>
<th>Display the endian set in the project. Display the same value as the value of the [Select endian] property of the build tool.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>The endian in the property of the build tool</td>
</tr>
<tr>
<td>How to change</td>
<td>Changes not allowed</td>
</tr>
</tbody>
</table>
**[System Configuration File Related Information] tab**

**Outline**
This tab shows the detailed information on the using system configuration file categorized by the following and the configuration can be changed.

- Realtime OS Generation Files
- Configurator Start Setting
- Service Call Information File

**Display image**

<table>
<thead>
<tr>
<th>Property</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Realtime OS Generation Files</td>
<td></td>
</tr>
<tr>
<td>Generate files</td>
<td>Yes (It updates the files when the cfg file is changed)</td>
</tr>
<tr>
<td>Output folder</td>
<td>%BuildNodeName%</td>
</tr>
<tr>
<td>Service Call Definition File name</td>
<td>kernel.sysin.h</td>
</tr>
<tr>
<td>System Information Header File name</td>
<td>kernel.idh</td>
</tr>
<tr>
<td>ROM Definition File name</td>
<td>kernel.romh</td>
</tr>
<tr>
<td>RAM Definition File name</td>
<td>kernel.ramh</td>
</tr>
<tr>
<td>System Definition File name</td>
<td>r600.inc</td>
</tr>
<tr>
<td>CMT Timer Definition File name</td>
<td>r1_cmt.h</td>
</tr>
<tr>
<td>Table File name</td>
<td>rtable.src</td>
</tr>
<tr>
<td>Configurator Start Setting</td>
<td></td>
</tr>
<tr>
<td>When undefined interrupt is generated, the interruption vs. Yes(-U)</td>
<td></td>
</tr>
<tr>
<td>The making situation of the file that the configurator generates vs. Yes(-V)</td>
<td></td>
</tr>
<tr>
<td>User options</td>
<td></td>
</tr>
<tr>
<td>Service Call Information File</td>
<td></td>
</tr>
<tr>
<td>The path that contains the service call information file</td>
<td>The path that contains the service call information file[O]</td>
</tr>
</tbody>
</table>

**Generate files**
Select whether to make a Realtime OS Generation Files which is output from a system configuration file. This file includes information of system initialization.
Explanation of each area

1) [Realtime OS Generation Files]
   The detailed information on the RI600PX generation files are displayed and the configuration can be changed.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate files</td>
<td>Select whether to generate realtime OS generation files and whether to update the realtime OS generation files when the system configuration file is changed.</td>
</tr>
<tr>
<td>Default</td>
<td>Yes (It updates the file when the .cfg file is changed)</td>
</tr>
<tr>
<td>How to change</td>
<td>Select from the drop-down list.</td>
</tr>
<tr>
<td>Restriction</td>
<td>Yes (It updates the file when the .cfg file is changed)</td>
</tr>
<tr>
<td>Restriction</td>
<td>No (It does not register the file to the project)</td>
</tr>
<tr>
<td>Restriction</td>
<td>Generates new realtime OS generation files and displays them on the project tree. If the system configuration file is changed when there are already realtime OS generation files, then realtime OS generation files are updated.</td>
</tr>
<tr>
<td>Restriction</td>
<td>Does not generate realtime OS generation files and does not display them on the project tree. If this item is selected when there are already realtime OS generation files, then the files themselves are not deleted.</td>
</tr>
</tbody>
</table>

| Output folder                   | Display the folder for outputting realtime OS generation files.              |
| Default                         | %BuildModeName%                                                             |
| How to change                   | Changes not allowed                                                         |

| Service Call Definition File Name | Display the name of the service call definition file that the cfg600px outputs. |
| Default                         | kernel_sysint.h                                                             |
| How to change                   | Changes not allowed                                                         |

| System Information Header File Name | Display the name of the system information header file that the cfg600px outputs. |
| Default                         | kernel_id.h                                                                  |
| How to change                   | Changes not allowed                                                         |

| ROM Definition File Name         | Display the name of the ROM definition file that the cfg600px outputs.       |
| Default                         | kernel_rom.h                                                                 |
| How to change                   | Changes not allowed                                                         |

| RAM Definition File Name         | Display the name of the RAM definition file that the cfg600px outputs.       |
| Default                         | kernel_ram.h                                                                 |
| How to change                   | Changes not allowed                                                         |

| System Definition File Name      | Display the name of the system definition file that the cfg600px outputs.    |
| Default                         | r600.inc                                                                     |
| How to change                   | Changes not allowed                                                         |

| CMT Timer Definition File Name   | Display the name of the CMT timer definition file which is generated by the cfg600px. |
| Default                         | ri_cmt.h                                                                     |
| How to change                   | Changes not allowed                                                         |
2) [Configurator Start Setting]

The start option of the configurator cfg600px can be specified.

<table>
<thead>
<tr>
<th>Table File Name</th>
<th>Display the name of the table file that the mkritblpx outputs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>ritable.src</td>
</tr>
<tr>
<td>How to change</td>
<td>Changes not allowed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>When undefined interrupt is generated, the interruption vector number is passed to system down routine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
</tr>
<tr>
<td>How to change</td>
</tr>
<tr>
<td>Restriction</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The making situation of the file that the configurator generates is displayed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
</tr>
<tr>
<td>How to change</td>
</tr>
<tr>
<td>Restriction</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
</tr>
<tr>
<td>How to change</td>
</tr>
<tr>
<td>Restriction</td>
</tr>
</tbody>
</table>

When an undefined interrupt occurs, the system down is caused. When -U option is specified, the vector number will be transferred to the system down routine (refer to "CAHPTER 15 SYSTEM DOWN"). This is useful for debugging. However, the kernel code size increases by about 1.5 kB.
3) [Service Call Information File]

Specify the path where the table generation utility mkritblpx retrieves the service call information files.

<table>
<thead>
<tr>
<th>The path that contains the service call information file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the service call information file (.mrc) or the path to the folder where the service call information files are retrieved. Note, when a folder path is specified, the sub folder is not retrieved. When relative path is specified, the project folder is the base folder. When absolute path is specified, the specified path is converted into the relative path which is based from the project folder. However, if the drive of the specified path is different from the drive of the project folder, this conversion is not done. Note, the project folder is passed to the mkritblpx regardless this setting. The following place holder can be specified. %BuildModeName% : Convert to the build mode name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How to change</th>
<th>Edit by the Path Edit dialog box which appears when clicking the […] button.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Up to 259 characters Note, when extension is not specified or the specified extension is not “.mrc”, the specified path is interpreted as folder.</td>
</tr>
</tbody>
</table>

Note 1  Refer to “2.6.1 Service call information files and “-ri600_preinit_mrc” compiler option” for the service call information file.

Note 2  When using the “optimization for accesses to external variables” compiler option, the CubeSuite+ generates the folder to store object files and service call information files for 1st build, and specifies this folder path for [Service Call Information File] tacit.

Note 3  The service call information files are generated to the same folder as object files at compilation. Please change this item appropriately when you do the operation to which the output folder of object files is changed.
APPENDIX B Floating-point Operation Function

It is only when the -fpu option is specified that the compiler outputs floating-point arithmetic instructions. If the -chkfpu option is specified in the assembler, the floating-point arithmetic instructions written in a program are detected as warning.

B.1 When Using Floating-point Arithmetic Instructions in Tasks and Task Exception Handling Routines

Make settings that include “FPSW” for Task context register (context) in System Information (system). As a result, the FPSW register is managed independently in each task.

The initial FPSW value for tasks and task exception handling routines is initialized by the value according to compiler options to be used. For details, refer to “4.2.4 Internal processing of task” and “6.2.2 Internal processing of task exception handling routine”.

B.2 When Using Floating-point Arithmetic Instructions in Handlers

It is necessary that the handler explicitly guarantee the FPSW register. The initial FPSW value of handlers is undefined.

To guarantee and initialize the FPSW register, write a program as follows.

```c
#include <machine.h>    // To use the intrinsic function get_fpsw() and set_fpsw(),
                        // include machine.h.
#include "kernel.h"
#include "kernel_id.h"

void handler (void) {
    unsigned long old_fpsw;    // Declare variable for saving the FPSW register
    old_fpsw = get_fpsw();      // Save the FPSW register
    set_fpsw (0x00000000);      // Initialize the FPSW register if necessary
    /* Floating-point arithmetic operation */
    set_fpsw (old_fpsw);       // Restore the FPSW register
}
```
When a MCU which support the DSP function is used, it is necessary to note the treatment of the ACC register (accumulator). Concretely, please note it as follows when you use the following DSP instructions which update ACC register.

- RXv1/RXv2 architecture common instruction
  MACHI, MACLO, MULHI, MULLO, RACW, MVTACHI, MVTACLO

- RXv2 architecture instructions
  EMACA, EMSBA, EMULA, MACLH, MSBHI, MSBLH, MSBLO, MULLH, MVTACGU, RACL, RDACL, RDACW

In no case does the compiler generate these instructions. Note also that if the -chkdsp option is specified in the assembler, the DSP function instructions written in a program are detected as warning.

**C.1 When Using DSP Instructions in Tasks and Task Exception Handling Routines**

Make settings that include “ACC” for Task context register (context) in System Information (system). As a result, the ACC register is managed independently in each task.

**C.2 When Using DSP Instructions in Handlers**

If the application contains any tasks or interrupt handlers that use the above-mentioned DSP instructions, it is necessary that all of the interrupt handlers guarantee the ACC register. There are the following two method.

1) Use “-save_acc” compiler option

2) Specify “ACC” for “pragma_switch” in all interrupt handler definition (Relocatable Vector Information (interrupt_vector[]) and Fixed Vector/Exception Vector Information (interrupt_fvector[])).
APPENDIX D  STACK SIZE ESTIMATION

If a stack overflows, the behavior of the system becomes irregular. Therefore, a stack must not overflow referring to this chapter.

D.1  Types of Stack

There are two types of stacks: the user stack and system stack. The method for calculating stack sizes differs between the user stack and system stack.

- **User stack**
  The stack used by tasks is called “User stack”.
  When a task is created by Task Information (task[]) in the system configuration file, the size and the name of the section where the stack is allocated are specified.
  When a task is created by cre_tsk or acre_tsk, the size and the start address of the user stack area are specified.

- **System stack**
  The system stack is used by handlers and the kernel. The system has only one system stack. The size is specified by System stack size (stack_size) in System Information (system). The section name of the system stack is “SI”.

D.2  Call Walker

The CubeSuite+ package includes “Call Walker” which is a utility tool to calculate stack size.

The Call Walker can display stack size used by each function tree.
D.3 User Stack Size Estimation

The quantity consumed of user stack for each task is a value in which the value calculated by the following expressions was rounded up to the multiple of 16.

\[
\text{Quantity consumed of user stack} = \text{treesz\_task} + \text{ctxsz\_task} + \text{treesz\_tex} + \text{ctxsz\_tex}
\]

- **treesz\_task**
  Size consumed by function tree that makes the task entry function starting point. (the size displayed by Call Walker).

- **treesz\_tex**
  Size consumed by function tree that makes the task exception handling routine entry function starting point. (the size displayed by Call Walker).

- **ctxsz\_task, ctxsz\_tex**
  Size for task context registers. The ctxsz\_task is for task, and ctxsz\_tex is for task exception handling routine. The size for task context registers is different according to the setting of Task context register (context) in System Information (system). Refer to Table D-1.

<table>
<thead>
<tr>
<th>Setting of system.context</th>
<th>Compiler option “-isa”</th>
<th>Size of Task Context Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>-</td>
<td>68</td>
</tr>
<tr>
<td>FPSW</td>
<td>-</td>
<td>72</td>
</tr>
<tr>
<td>ACC</td>
<td>“-isa=rxv2”</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>“-isa=rxv1” or not specify “-isa”</td>
<td>76</td>
</tr>
<tr>
<td>FPSW,ACC</td>
<td>“-isa=rxv2”</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>“-isa=rxv1” or not specify “-isa”</td>
<td>80</td>
</tr>
<tr>
<td>MIN</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>MIN,FPSW</td>
<td>-</td>
<td>48</td>
</tr>
<tr>
<td>MIN,ACC</td>
<td>“-isa=rxv2”</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>“-isa=rxv1” or not specify “-isa”</td>
<td>52</td>
</tr>
<tr>
<td>MON,FPSW,ACC</td>
<td>“-isa=rxv2”</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>“-isa=rxv1” or not specify “-isa”</td>
<td>56</td>
</tr>
</tbody>
</table>

Note: Compiler option “-isa” is supported by the compiler CC-RX V2.01 or later.
D.4 System Stack Size Estimation

The system stack is most consumed when an interrupt occurs during service call processing followed by the occurrence of multiple interrupts. The quantity consumed of system stack is calculated by the following expressions.

\[
\text{Quantity consumed of system stack} = svcsz + \sum_{k=1}^{15} \text{inthdrsz}_k + \text{sysdwnsz}
\]

- **svcsz**
  The maximum size among the service calls to be used in the all processing program. The value \( svcsz \) depends on the RI600PX version. For details, refer to release notes.

- **inthdrsz**
  Size consumed by function tree that makes the interrupt handler entry function starting point. (the size displayed by Call Walker).
  The "k" means interrupt priority level. If there are multiple interrupts in the same priority level, the \( \text{inthdrsz}_k \) should select the maximum size among the handlers.
  The size used by the base clock interrupt handler (the interrupt priority level is specified by Base clock interrupt priority level (IPL) in Base Clock Interrupt Information (clock)) is the maximum value in the following Please refer to the release notes for \( \text{clocksz1} \), \( \text{clocksz2} \) and \( \text{clocksz3} \).
  Don't have to add the size used by the base clock interrupt handler when base clock timer is not used (clock.timer = NOTIMER).
  
  - \( \text{clocksz1} + \text{cycsz} \)
  - \( \text{clocksz2} + \text{almsz} \)
  - \( \text{clocksz3} \)
    - **cycsz**
      Size consumed by function tree that makes the cyclic handler entry function starting point. (the size displayed by Call Walker).
      If there are multiple cyclic handlers, the \( \text{cycsz} \) should select the maximum size among the handlers.
    - **almsz**
      Size consumed by function tree that makes the alarm handler entry function starting point. the size displayed by Call Walker).
      If there are multiple alarm handlers, the \( \text{cycsz} \) should select the maximum size among the handlers.

- **sysdwnsz**
  Size consumed by function tree that makes the system down routine entry function starting point. (the size displayed by Call Walker) + 40. When the system down routine has never been executed, \( \text{sysdwnsz} \) is assumed to be 0.
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Apr 01, 2012</td>
<td>-</td>
<td>First Edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Sep 20, 2013 (RI600PX V1.02.00)</td>
<td>24</td>
<td>&quot;2.6.2 Compiler option for the boot processing file&quot; has been detailed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>With support of RXv2 architecture, the composition of kernel libraries have been changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28, 217, 225, 527, etc.</td>
<td>With support of RXv2 architecture, the explanation about FIX_INTERRUPT_VECTOR section and EXTB register have been added or changed. Moreover, &quot;fixed vector&quot; has been replaced by &quot;fixed vector/exception vector&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>&quot;2.6.5 Initialized data section&quot; has been added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42, 67, 557</td>
<td>The specification of FPSW register when task and task exception handling routine processing is started has been changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>217</td>
<td>With support of RXv2 architecture, the explanation about compiler option &quot;-isa&quot; and &quot;-cpu&quot; have been added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>221, 456</td>
<td>The explanation about starting of RI600PX has been improved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>222</td>
<td>Expression of section &quot;17.4 Section Initialization Function (_INTSCT())&quot; has been improved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>230</td>
<td>With revision to V1.02.00, the definition value of TKERNEL_PRVER has been changed into 0x0120.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>485</td>
<td>With support of RXv2 architecture, Table 20-2 has been changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>489</td>
<td>The explanation of Table 20-7 has been improved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>501</td>
<td>With support of RXv2 architecture, Table 20-8 has been changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>558</td>
<td>The RXv2 instructions have been added to DSP instructions which update ACC register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>558</td>
<td>The description &quot;All interrupt handlers explicitly guarantee the ACC register&quot; has been deleted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>560</td>
<td>With support of RXv2 architecture, Table D-1 has been changed.</td>
</tr>
</tbody>
</table>