To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

RENESAS

User's Manual

RX850 Pro Ver. 3.30

Real-Time Operating System

Coding for CubeSuite

Target Tool
RX850 Pro Ver.3.30

Document No. U19429EJ1V0UM00 (1st edition)
Date Published December 2008

[MEMO]

SUMMARY OF CONTENTS

[MEMO]

INTRODUCTION

Readers	This manual is intended for users who design and develop application systems using V850 microcontrollers products.
Purpose	This manual is intended for users to understand the functions of RX850 Pro described the organization listed below.
Organization	This manual consists of the following major sections.

- OVERVIEW
- SYSTEM CONSTRUCTION
- NUCLEUS
- TASK MANAGEMENT FUNCTION
- SYNCHRONOUS COMMUNICATION FUNCTIONS
- INTERRUPT MANAGEMENT FUNCTION
- MEMORY POOL MANAGEMENT FUNCTION
- TIME MANAGEMENT FUNCTION
- SCHEDULER
- SYSTEM INITIALIZATION
- INTERFACE LIBRARY
- SYSTEM CALLS
- SYSTEM CONFIGURATION FILE
- CONFIGURATOR (CF850 Pro)

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, microcontrollers, C language, and assemblers.

To understand the hardware functions or instruction functions of the V850 microcontrollers
→ 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
Numerical representation:	Binary...XXXX or XXXXB
	Decimal...XXXX
	Hexadecimal...0xXXXX

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

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

Related Documents Read this manual together with the following documents.
The related documents indicated in this publication may include preliminary versions.
However, preliminary versions are not marked as such.

Documents related to development tools (user's manuals)

Document Name		Document No.
RX Series	Start for CubeSuite	U19428E
	Message for CubeSuite	U19433E
RX850 Pro Ver.3.30	Coding for CubeSuite	This document
	Debug for CubeSuite	U19431E
	Analysis for CubeSuite	U19432E
	Internal Structure for CubeSuite	U19434E
CubeSuite Integrated Development Environment	Start	U19549E
	Programming	U19390E
	Message	U19550E
	V850 Coding	U19383E
	V850 Build	U19386E
	V850 Debug	U19389E
	V850 Design	U19380E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 OVERVIEW

Rapid advances in semiconductor technologies have led to the explosive spread of microprocessors such that they are now to be found in more fields than many would have imagined only a few years ago. In line with this spread, the number of processing programs that must be created for microprocessors is also increasing. This rule of growth makes it difficult to create processing programs specific to given hardware.

For this reason, there is a need for operating systems (OSs) that can fully exploit the capabilities of the latest generation of ever-newer high-performance, multi-function microprocessors.

Operating systems are broadly classified into 2 types: program-development OSs and control OSs. Program-development OSs are to be found in those environments in which standard OSs (e.g., Windows®) predominate because the hardware configuration to be used for development can be limited to some extent (e.g., personal computers).

Conversely, control OSs are incorporated into control units. That is, these OSs are found in those environments where standard OSs cannot easily be applied because the hardware configuration varies from system to system and because efficient operation matching the application is required.

To satisfy these demands, NEC Electronics has developed and released not only the V850 microcontrollers but also the RX850 Pro operating system, which allows users to fully exploit the functions of these microcontrollers and support systematic software creation.

The RX850 Pro is a control OS for real-time, multitasking processing; it has been developed to increase the application range of high-performance, multi-function microprocessors and further improve their versatility.

1.1 Outline

The RX850 Pro is an embedded real-time, multitask control OS that provides a highly efficient real-time, multitasking environment to increase the application range of processor control units.

The RX850 Pro is a high-speed, compact OS capable of being stored in and run from the ROM of a target system.

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 operating systems have 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 Multitask OS

A "task" is the minimum unit in which a program can be executed by an OS. "Multitasking" 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 multitask OS enables the parallel processing of tasks by switching the tasks to be executed as determined by the system.

One important purpose of a multitask OS is to improve the throughput of the overall system through the parallel processing of multiple tasks.

1.2 Applications

The RX850 Pro is suitable for the following devices.

- Systems using motor controllers
  PPCs, printers, FAXes

- Systems requiring low power consumption
  Cellular phones, personal handyphones (PHS), digital still cameras

CHAPTER 2 SYSTEM CONSTRUCTION

This chapter explains how to construct an application system using the RX850 Pro.

2.1 Outline

System construction involves incorporating created load modules into a target system, using the file group copied from the RX850 Pro distribution media to the user development environment (host machine).

The system construction procedure is outlined below.

1) Creating System Configuration File

2) Creating System Initialization Processing

- Boot processing
- Hardware initialization module
- Initialization handler
- Interrupt entry

3) Creating Processing Programs

- Tasks
- Directly Activated Interrupt Handler
- Indirectly Activated Interrupt Handler
- Cyclic Handler
- Extended SVC Handler

Remark The programs are created by using C language or assembly language.

4) Creating Initialization Data Save Area

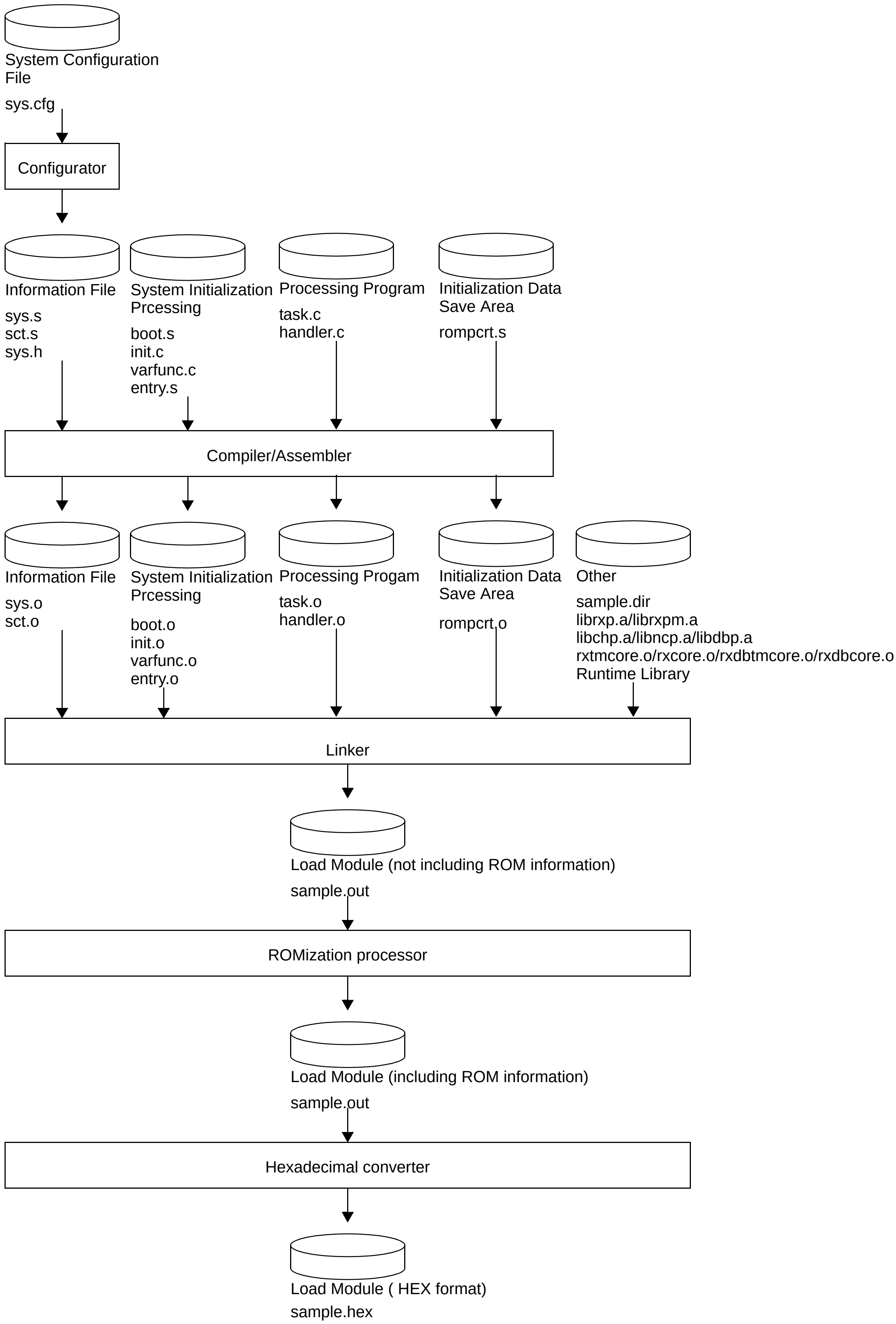
5) Creating Llink Directive File

6) Creating Load Module

7) Embedding System

Figure 2-1 shows the procedure for organizing the system.

Figure 2-1 Example of System Construction



The flow of organizing the system is explained based on the sample program supplied with the CA850.
The program is stored in the following folder if the RX850 Pro has been installed in the folder <rx_root>.

<rx_root>\smp850e\rx85p\src

2.2 Creating System Configuration File

Create an information table, called a system configuration file, which holds the various data used with the RX850 Pro. This file is necessary for creating the following using the configurator.

- System information table file
- System call table file
- System information header file

The "system information table file" contains information on the resources of the RX850 Pro, such as tasks, semaphores, and memory pools. The "system call table file" contains a list of system calls used for applications. The "system information header file" has a description that makes the symbol names specified as resource IDs, such as those of tasks and semaphores created with the system information table file, correspond to the actual symbol ID numbers, by using the #define instruction.

The sample system configuration file is:

- sys.cfg

For the contents and syntax of the system configuration file, see "CHAPTER 13 SYSTEM CONFIGURATION FILE".

2.3 Creating System Initialization Processing

The system initialization processing is a function consisting of program segments that are dependent upon the user's target system. This function is used to facilitate porting and customization.

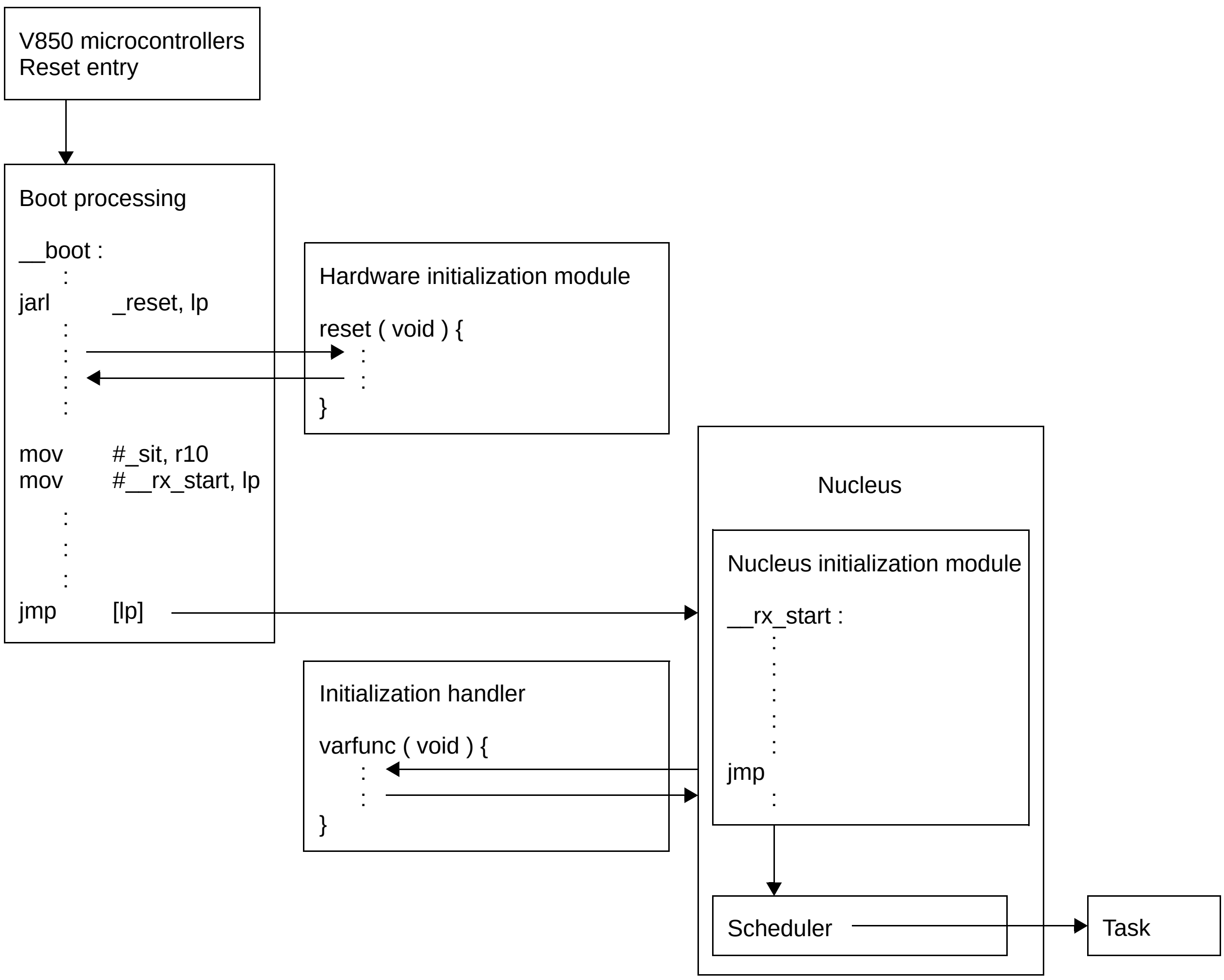
The sample file is as follows.

Table 2-1 Configuration of System Initialization Processing

Sample File Name	Type	Function Name	Role
boot.s	Boot processing	_boot	Boot processing of system
init.c	Hardware initialization module	reset	Initialization processing of hardware
varfunc.c	Initialization handler	varfunc	Initialization processing of software
entry.s	Interrupt entry	None	Processing to branch to interrupt servicing

The rough flow of the system initialization processing is illustrated below. Each processing is explained next.

Figure 2-2 Flow of System Initialization Processing



2.3.1 Boot processing

The boot processing is assigned to the reset entry (handler address: 0x0) of the V850 microcontrollers and is the system initialization processing that is executed first.

The description following the label "_boot" in the sample file boot.s is the entity of the boot processing. The instructions that cause execution to jump from the reset entry to this label are as follows. These instructions are in the same entry.s file.

```
.section     "RESET"
.extern      __boot

mov          #__boot, lp
jmp          [lp]
```

The lower 2 lines of the instructions are assigned to the handler address [0x0]. When reset is executed, therefore, these instructions are executed, execution jumps to _boot, and the boot processing is executed.

The following must be performed as part of the boot processing.

1) Setting of stack pointer (sp) used in boot processing
2) Setting of text pointer (tp) and global pointer (gp)
3) Setting of symbol _sit to r10 register address
4) Setting of symbol __rx_start to lp register
5) Issuance of jmp instruction to transfer control to nucleus initialization module

In addition to the above, processing (jarl_reset, lp) that causes execution to jump to the reset function, which is a "hardware initialization module", is executed between 3) and 4) in the sample.

The stack pointer to be set in 1) above is independent of the stack for tasks and interrupt handlers. After the RX850 Pro has been started, the stack used by tasks and interrupt handlers is managed by the RX850 Pro itself, by using the system information table file, and the stack pointer is automatically switched by means of task switching or interrupts. Therefore, the stack pointer specified in the boot processing is used before the RX850 Pro is started.

This stack pointer is used, for example, when execution jumps to a function and that function has data to be saved to the stack. This stack pointer is used if it is necessary to use the stack with the reset function of the sample.

Although the sample program uses a stack of 0x28000 bytes, such a high-capacity stack is not usually necessary. This stack area is of the size defined by the system memory area used for the RX850 Pro ("System memory information" in the system configuration file), and is used as the system memory area after the RX850 Pro has been started.

In the sample, the bss area on RAM is initialized (cleared to 0) in boot processing.

The default value data is copied by creating an area of the default value data (rompcrt.s) and by using the function _rcopy. See CubeSuite V850 Build User's Manual for details.

At the end of the boot processing, processing for 4) to 6) is necessary. Perform the following processing.

```
.extern      _sit
mov          #_sit, r10

.extern      __rx_start
mov          #__rx_start, lp
jmp          [lp]
```

The description in the RX850 Pro following the symbol "__rx_start" is the nucleus initialization processing of the RX850 Pro. After the boot processing has been completed, transfer control to the nucleus initialization processing by using the jmp instruction. At this time, substitute the address of _sit symbol into the r10 register. This is because resources are created and initialized based on this address substituted into the r10 register and the "system information table file" created from the system configuration file.

NEC Electronics recommends changing the description of the boot processing to an environment suitable for the user, based on the boot processing of the sample.

2.3.2 Hardware initialization module

The hardware initialization module consists of functions called from the boot processing in the sample program. These functions are provided to initialize the hardware on the target system, as a series of boot processing.

In the sample program, the reset function initializes the hardware. This function is called from the boot processing. There is no problem even if this function is not used, if initializing the hardware is not necessary or if the hardware is initialized by other processing.

The hardware initialization module of the sample program performs the following processing.

1) Initialization of interrupt controller/clock controller

2) Initialization of peripheral I/O register/controller

3) Returning control to boot processing

2.3.3 Nucleus initialization module

The nucleus initialization module is an internal routine of the RX850 Pro that is executed after completion of the boot processing. This module creates the RX850 Pro system management block, and creates and initializes information on items such as tasks, semaphores, and memory pools, based on the "system information table file" created from the system configuration file.

The RX850 Pro places the CPU in the HALT status if initialization was not correctly performed by the nucleus initialization module. If the RX850 Pro is not started and the CPU enters the HALT status after execution has jumped from the boot processing to the initialization processing, the system memory area ("System memory information" in the system configuration file) used to create the management block of the RX850 Pro is probably insufficient. Check that a sufficient memory capacity has been reserved.

Once initialization has been completed in the nucleus initialization module, an initialization handler is called. This initialization handler is specified by "Initialization handler information" of the system configuration file and is the varfunc function in the sample program. For details of this function, see "2.3.4 Initialization handler".

When control has returned from the initialization handler, the scheduler is started, and then the RX850 Pro is started.

2.3.4 Initialization handler

The initialization handler is a function that is called from the nucleus initialization module. Describe the processing to be performed before starting the RX850 Pro.

The initialization handler calls a function specified by "Initialization handler information" of the system configuration file. In the sample program, this function is varfunc. Even if the processing is not necessary, create the function as a function that performs no processing. At the end of the handler, return to the nucleus initialization processing by using the return instruction.

In the sample program, this function is defined as a function that executes nothing. The default data value can be copied into the initialization handler. See CubeSuite V850 Build User's Manual for details about how to copy the default value data.

Remark1 When passing control from the nucleus initialization module to the initialization handler, the RX850 Pro switches the current stack to the system stack that is specified in System information during configuration.

Remark2 When passing control from the nucleus initialization module to the initialization handler, the RX850 Pro switches the values of the text pointer (tp) and global pointer (gp) to values that are defined in Initialization handler information during configuration.

Remark3 The RX850 Pro performs no operations on the element pointer (ep). The ep value used during the initialization handler processing therefore differs from the value set during boot processing.

Remark4 The initialization handler is called before the RX850 Pro completes all of the initialization processing. Therefore, if interrupts for the initialization handler are enabled or a system call is issued by the initialization handler, the operation is not guaranteed.

2.3.5 Interrupt entry

An interrupt entry is an instruction that is executed if an interrupt occurs, and is assigned to the "interrupt handler address" of the V850 microcontrollers. The interrupt entry must be defined for all the interrupts used by the user, and must be described in assembly language. The interrupt handler of the sample is described in "entry.s".

The interrupts of the RX850 Pro are handled by 2 types of handlers: a "directly activated interrupt handler" and an "indirectly activated interrupt handler", each of which differs from the other in entry description.

In the directly activated interrupt handler, describe a branch instruction in the same manner as an ordinary interrupt entry. In the sample program (for V850ES/V850E1/V850E2 core), the interrupt "INTP110 (handler address: 0x180)" is an example of a directly activated interrupt handler.

The .section quasi directive is used. See CubeSuite V850 Build User's Manual for details about each instruction.

The entry of the directly activated interrupt handler is as follows.

```
.section     "INTP110"
jr           _intp110_entry
```

The destination label "_intp110_entry" is defined in the same file, and execution jumps to the entity of the handler (intp130) after the preprocessing and post-processing of the directly activated interrupt handler are described (macro description).

The indirectly activated interrupt handler uses the macro provided for the RX850 Pro. This means that the contents of the macro must be assigned to the handler address. The macro name is "RTOS_IntEntry_Indirect". In the sample program (for V850ES/V850E1/V850E2 core), the interrupt "INTP120 (handler address: 0x1c0)" is used as an example of an indirectly activated interrupt handler.

The .section quasi directive is used. See CubeSuite V850 Build User's Manual for details about each instruction.

The entry of the indirectly activated interrupt handler is as follows.

```
.section     "INTP120"
RTOS_IntEntry_Indirect
```

An interrupt entry must also be registered for the clock interrupt used with the RX850 Pro in the same manner as the indirectly activated interrupt handler. This interrupt entry is described in the sample program (for V850ES/V850E1/V850E2 core,) as follows because "INTCMD0 (handler address: 0x240)" is used as a clock interrupt.

```
.section     "INTCMD0"
RTOS_IntEntry_Indirect
```

Remark For the interrupt handler defined in Indirectly activated interrupt handler information and the clock handler that corresponds to the interrupt source numbers of the timer defined in System information, the configurator automatically outputs the relevant interrupt entry to the system information table, so the user is not required to write the relevant interrupt entry.

If -ne is specified as the configurator start option, output of the interrupt entry to the system information table is suppressed.

2.4 Creating Processing Programs

Create a processing program (application).
The application processing units required for the RX850 Pro are broadly classified into the following.

- Tasks
- Directly Activated Interrupt Handler
- Indirectly Activated Interrupt Handler
- Cyclic Handler
- Extended SVC Handler

The contents of the sample, except the extended SVC handler, are shown below. The following table shows the files included in the sample program for V850 cores.

Table 2-2 Configuration of Processing Program

Sample File Name	Type	Function Name	Role
task.c	Task	task1 task2	Task entity
handler.c	Indirectly activated interrupt handler Directly activated interrupt handler Cyclic handler	inthdr1 inthdr2 cychdr1 cychdr2	Handler processing

If a processing program described in C issues a system call, include the header file "stdrx85p.h" supplied by the RX850 Pro. This file contains the definition necessary for using the system call. The header file "usr.h" in the sample program defines constants used by functions as necessary and is included in the program. In the sample program, only a macro of constants is defined.

2.5 Creating Initialization Data Save Area

It is necessary to create an area for saving the initialization data. This is because it is necessary to store the initialization data in ROM and to copy the default values of the data to RAM before executing a program. Creating a saving area for the initialization data involves reserving a ROM area in which the initialization data is to be stored.

See the description of "ROMization processor" in CubeSuite V850 Build User's Manual for details about how to create this area.

2.6 Creating Llink Directive File

Create a link directive file (section map file) containing the "section information" and "address information" referenced by the linker when it links modules. The following sample file is a link directive file.

The following file of the sample is the link directive file.

- sample.dir

The sections listed in the table below are essential for the RX850 Pro.

Table 2-3 Essential Sections for RX850 Pro

Section Name	Type of Area	Remark
.sit	System information area	Essential
.system	RX850 Pro system call location area	Essential
.system_cmn	RX850 Pro scheduler-related area	Essential
.system_int	RX850 Pro interrupt servicing-related area	Essential
.text	RX850 Pro interface library location area	Essential
Any	Area in which system memory area System Memory Pool 0 is assigned	Essential
Any	Area in which system memory area System Memory Pool 1 is assigned	Can be omitted
Any	Area in which system memory area User Memory Pool 0 is assigned	Can be omitted
Any	Area in which system memory area User Memory Pool 1 is assigned	Can be omitted

".sit" has a const attribute, 4 system memory areas have a bss attribute, and the other sections have a text attribute. Information on these sections must be defined in the link directive file (section map file). As described in the file of the sample, these sections define position information.

The file of the sample (for V850 core) that corresponds to these sections is as follows.

```
    :
    :
CONST   : !LOAD ?R      V0x00002000 {
        .sit            = $PROGBITS ?A  .sit;
        .const          = $PROGBITS ?A  .const;
};
TEXT    :!LOAD       ?RX {
        .pro_epi_rutime = $PROGBITS ?AX .pro_epi_runtime;
        .system         = $PROGBITS ?AX .system;
        .system_cmn     = $PROGBITS ?AX .system_cmn;
        .system_int     = $PROGBITS ?AX .system_int;
        .text           = $PROGBITS ?AX .text;
};
S0MEMA  :!LOAD  ?RW     V0xffffc000 {
        .spol0area      = $NOBITS   ?AW .spol0area;
};
S1MEMA  :!LOAD  ?RW     V0xffffcc00 {
        .spol1area      = $NOBITS   ?AW .spol1area;
};
U0MEMA  :!LOAD  ?RW     V0xffffd200 {
        .upol0area      = $NOBITS   ?AW .upol0area;
};
    :
    :
```

In addition, define sections related to the RAM area, such as .data/.bss section, and those related to the ROM area, such as the const section, as necessary. NEC Electronics recommends changing the description of the link directive file (section map file) to an environment suitable for the user.

See CubeSuite V850 Build User's Manual for details about how to describe the link directive file.

2.7 Creating Load Module

Run a build on CubeSuite for files created in sections from "2.2 Creating System Configuration File" to "2.6 Creating Llink Directive File", and library files provided by the RX850 Pro and CA850, to create a load module.

1) Create or load a project

Create a new project, or load an existing one.

Note See RX Series Start User's Manual or CubeSuite Start User's Manual for details about creating a new project or loading 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.

Note See CubeSuite V850 Build User's Manual for details about setting the active project.

3) Set build target files

For the project, add or remove build target files and update the dependencies.

Note See CubeSuite V850 Build User's Manual 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.

- System configuration file created in "2.2 Creating System Configuration File"

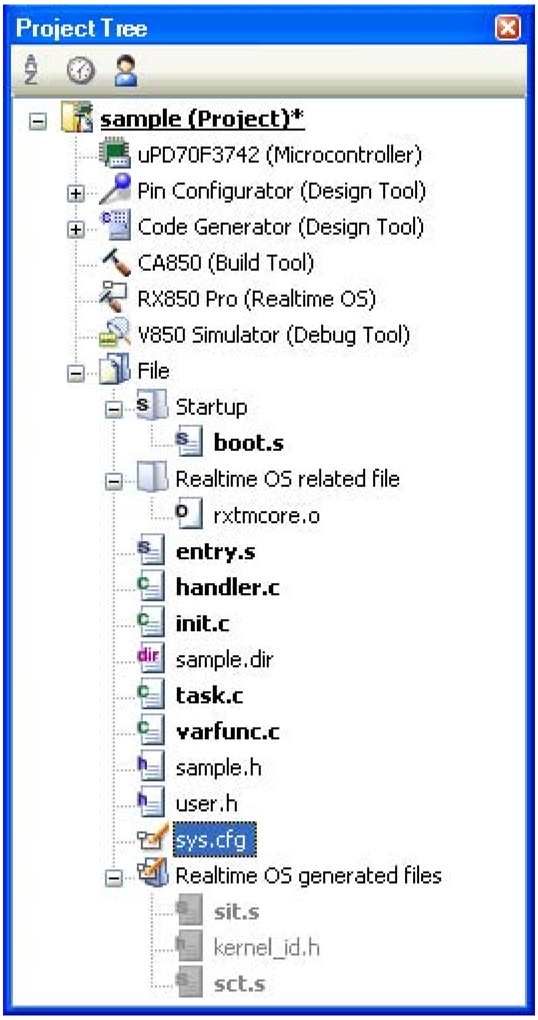
  Note Specify "cfg" as the extention of the system configuration file name.
  If the extension is different, "cfg" is automatically added (for example, if you designate "aaa.c" as a file name, the file is named as "aaa.c.cfg").

- C/assembly language source files created in "2.3 Creating System Initialization Processing"
- C/assembly language source files created in "2.4 Creating Processing Programs"
- C/assembly language source files created in "2.5 Creating Initialization Data Save Area"
- Link directive file created in "2.6 Creating Llink Directive File"

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 table file
- System information header file
- System call table file

Figure 2-3 Project Tree Panel (After Adding sys.cfg)



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.

4) Specify library and object linking

When linking applications that use the RX850 Pro, the following libraries must be referenced and objects must be linked.

- Nucleus library

librxp.a:	Immediately before rel_blk is issued, the first 4 bytes of the target memory block need to be cleared to 0.
librxpm.a:	Immediately before rel_blk is not issued, the first 4 bytes of the target memory block need to be cleared to 0.

- Interface library

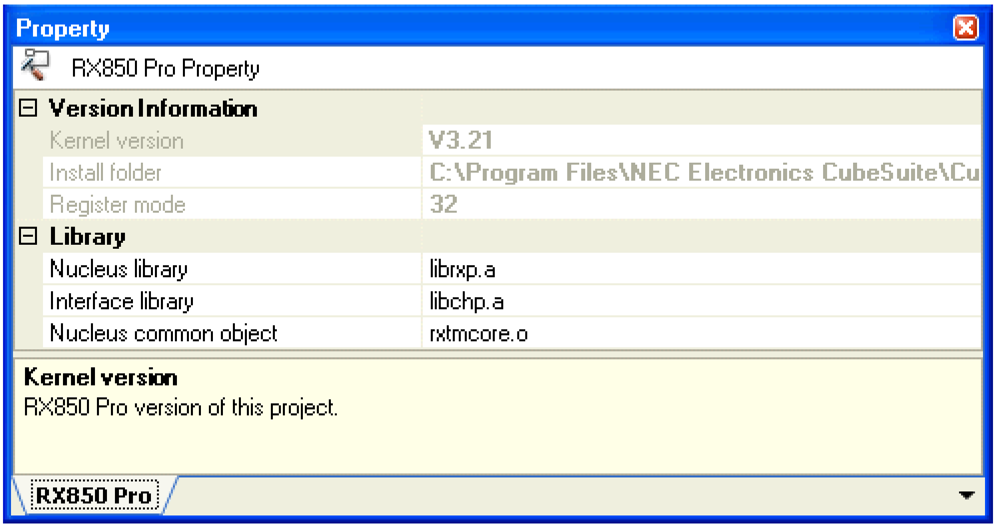
libchp.a:	With parameter check
libncp.a:	Without parameter check

- Nucleus common object

rxtmcore.o:	In the cyclic handler, the acceptance of maskable interrupts that have higher priority than clock interrupts is enabled.
rxcore.o:	In the cyclic handler, the acceptance of all maskable interrupts is enabled.

Select the Realtime OS node on the project tree to open [RX850 Pro] tab on the Property panel.
Set the nucleus library, interface library, and nucleus common object linking on each property.

Figure 2-4 Property Panel: [RX850 Pro] Tab

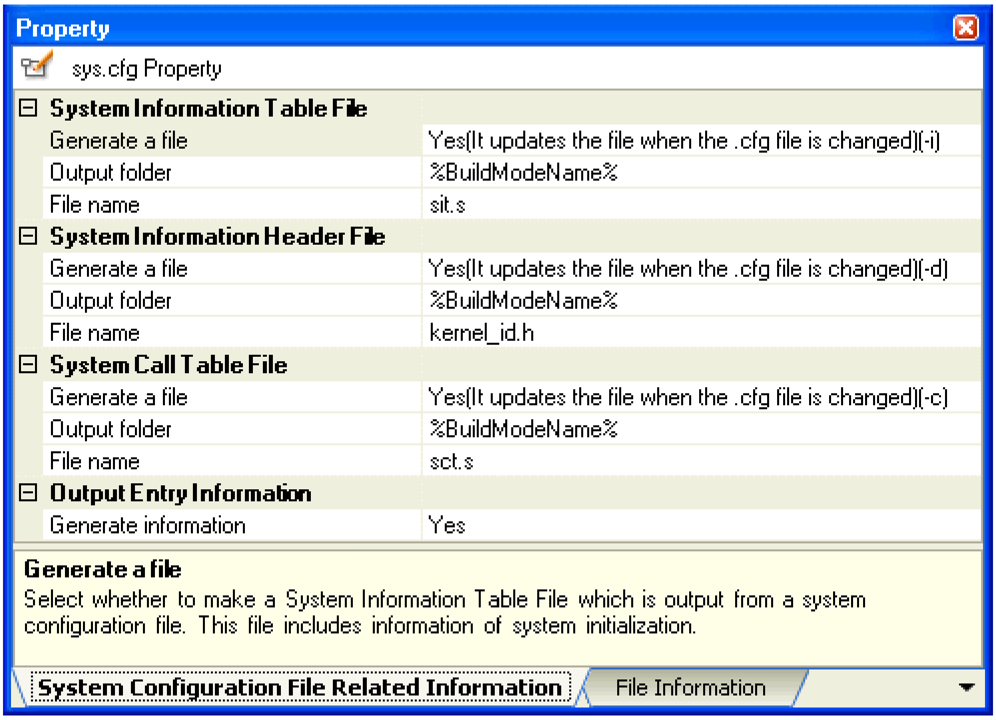


Two types of interface libraries and 2 types of nucleus common objects are available. Which type is to be used should be determined according to the application. For details of the nucleus library, see "CHAPTER 11 INTERFACE LIBRARY". For details of the nucleus common object, refer to "CHAPTER 8 TIME MANAGEMENT FUNCTION".

5) Set the output of information 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 information files (system information table file, system information header file, and system call table file).

Figure 2-5 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.

Note See CubeSuite V850 Build User's Manual for details about specifying the output of a load module file.

7) Set build options

Set the options for the compiler, assembler, linker, and the like.

Note 1 See CubeSuite V850 Build User's Manual for details about setting build options.

Note 2 The following table lists the compiler option specifications that are prohibited.

Table 2-4 Options Prohibited for Setting

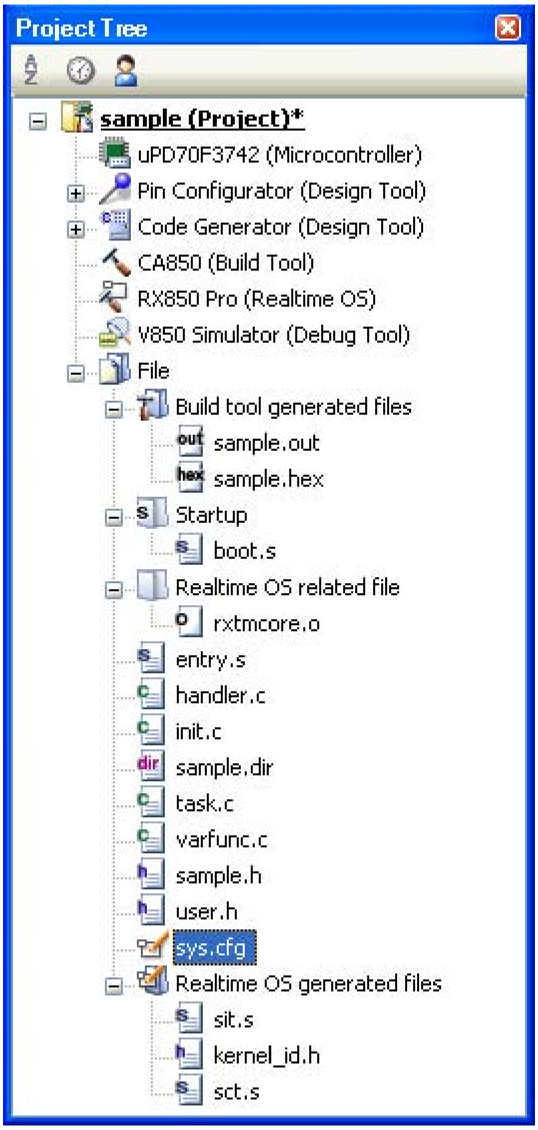
Compiler Option	Remark
-reg22 -reg26	Specification of these options is prohibited because the RX850 Pro only supports the 32-register mode.
-Xpack=1 -Xpack=2	Specification of these options is prohibited because the RX850 Pro performs processing on the assumption that data structures are assigned to areas with 4-byte alignment.

8) Run a build

Run a build to create a load module.

Note See CubeSuite V850 Build User's Manual for details about runnig a build.

Figure 2-6 Project Tree Panel (After Running Build)



The load module file created by the linker correctly locates the initialization data in RAM. If initialization data exists in the application, a module that reserves an initialization data saving area and that incorporates a copy routine must be created. In this case, a load module that passes through a ROMization processor must be created for the load module created by the linker.

See CubeSuite V850 Build User's Manual for details about how to use the ROMization processor and for details of the copy routine.

9) Save the project

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

Note See CubeSuite Start User's Manual for details about saving the project.

2.8 Embedding System

If the output of a hex file are set in 6) of "2.7 Creating Load Module", a hex file is created.
Then, embed this file into the system by using a ROM writer, etc.

CHAPTER 3 NUCLEUS

This chapter describes the nucleus, which is the core of the RX850 Pro.

3.1 Outline

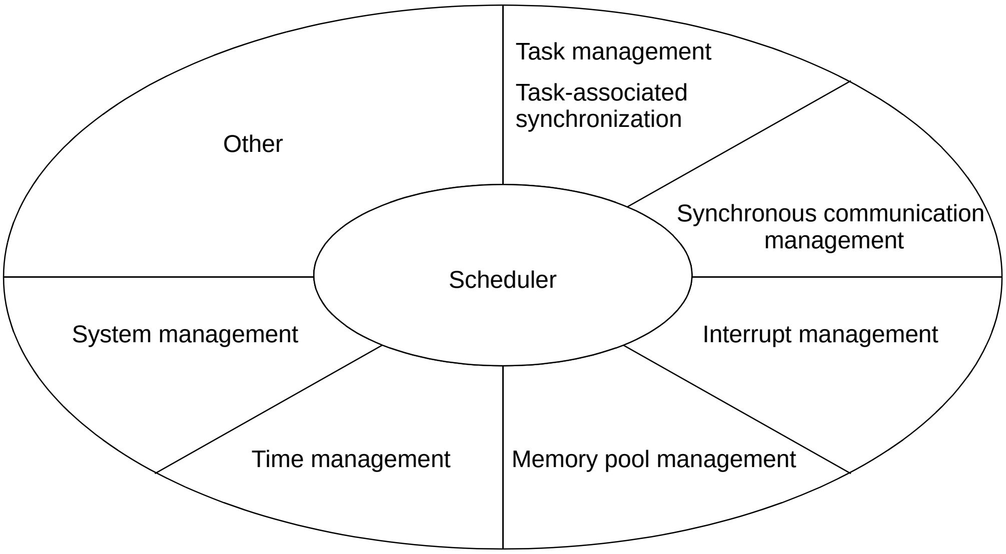
The nucleus forms the heart of the RX850 Pro, a system that supports real-time, multitask control. The nucleus provides the following functions.

- Creation/initialization of management objects
- Processing of system calls issued by processing program (task/non-task)
- Selection of the processing program (task/non-task) to be executed next, according to an event that occurs internal or external to the target system

Management object creation/initialization and system call processing are executed by management modules. Program selection is performed by a scheduler.

The configuration of the RX850 Pro nucleus is shown below.

Figure 3-1 Nucleus Configuration



3.2 Functions

The nucleus consists of various kinds of management modules and a scheduler.
This section outlines the functions of the management modules and scheduler.
See "CHAPTER 4 TASK MANAGEMENT FUNCTION" through "CHAPTER 9 SCHEDULER" for details of the individual functions.

- Task management function
  This module manipulates and manages the states of a task, the minimum unit in which processing is performed by the RX850 Pro. For example, the module can create, start, run, stop, terminate, and delete a task.

- Synchronous communication function
  This module enables 3 functions related to synchronous communication between tasks: exclusive control, wait, and communication.

Exclusive control function:	Semaphore
Wait function:	Eventflag
Communication function:	Mailbox

- Interrupt management function
  This module executes the processing related to maskable interrupts, such as the registration of an indirectly activated interrupt mask, return from a directly activated interrupt handler, and change or acquisition of the interrupt-enable level.

- Memory pool management function
  This module manages the memory area specified at configuration, dividing it into the following 2 areas.

  - RX850 Pro area
    Management objects
    Memory pool

  - Processing program (task/non-task) area
    Text area
    Data area
    Stack area

  The RX850 Pro also applies dynamic memory pool management. For example, the RX850 Pro provides a function for acquiring and returning a memory area to be used as a work area as required.
  By exploiting this ability to dynamically manage memory, the user can utilize a limited memory area with maximum efficiency.

- Time management function
  This module supports a timer operation function (such as delayed wake-up of a task or activation of a cyclic handler) that is based on clock interrupts generated by hardware (such as a clock controller).

- Scheduler
  By monitoring the dynamically changing states of tasks, this module manages and determines the order in which tasks are executed and optimally assigns tasks a processing time.
  The RX850 Pro determines the task execution order according to assigned priority levels and by applying the FCFS method. When started, the scheduler determines the priority levels assigned to the tasks, selects an optimum task from those ready to be executed (run or ready state), and optimally assigns tasks a processing time.

  Remark In the RX850 Pro, the smaller the value of the priority assigned to the task, the higher the priority.

CHAPTER 4 TASK MANAGEMENT FUNCTION

This chapter describes the task management function performed by the RX850 Pro.

4.1 Outline

Tasks are execution entities of arbitrary sizes, making them difficult to manage directly. The RX850 Pro manages task states and tasks themselves by using management objects that correspond to tasks on a one-to-one basis.

Remark A task uses the execution environment information provided by the program counter, work registers, and the like when it executes processing. This information is called the task context. When the task execution is switched, the current task context is saved and the task context for the next task is loaded.

4.2 Task States

The task changes its state according to how resources required to execute the processing are acquired, whether an event occurs, and so on.
The RX850 Pro classifies task states into the following 7 types.

- Non-existent state
  A task in this state has not been created or has been deleted.
  A task in the non-existent state is not managed by the RX850 Pro even if its execution entity is located in memory.

- Dormant state
  A task in this state has just been created or has already completed its processing.
  A task in the dormant state is not scheduled by the RX850 Pro.
  This state differs from the wait state in the following points:

  - All resources are released.
  - The task context is initialized when the processing is resumed.
  - A state manipulation system call (ter_tsk, chg_pri, etc.) causes an error.

- Ready state
  A task in this state is ready to perform its processing. This task has been waiting for a processing time to be assigned because another task having a higher (or the same) priority level is being executed.
  A task in the ready state is scheduled by the RX850 Pro.

- Run state
  A task in this state has been assigned a processing time and is currently performing its processing.
  Within the entire system, only a single task can be in the run state at any one time.

- Wait state
  A task in this state has been stopped because the requirements for performing its processing are not satisfied.
  The processing of this task is resumed from the point at which it was stopped, so the values that were being used immediately before the stop are restored to the task context required to resume the processing.
  The RX850 Pro further divides tasks in the wait state into the following 6 groups, according to the conditions which caused the transition to the wait state.

  Wake-up wait state: A task enters this state if the counter for the task (registering the number of times the wake-up request has been issued) indicates 0x0 upon the issuance of slp_tsk or tslp_tsk.

  Resource wait state: A task enters this state if it cannot acquire a resource from the relevant semaphore upon the issuance of wai_sem or twai_sem.

  Eventflag wait state: A task enters this state if a relevant eventflag does not satisfy a predetermined condition upon the issuance of wai_flg or twai_flg.

Message wait state:	A task enters this state if it cannot receive a message from the relevant mailbox upon the issuance of rcv_msg or trcv_msg.
Memory block wait state:	A task enters this state if it cannot acquire a memory block from the relevant memory pool upon the issuance of get_blk or tget_blk.
Timeout wait state:	A task enters this state upon the issuance of dly_tsk.

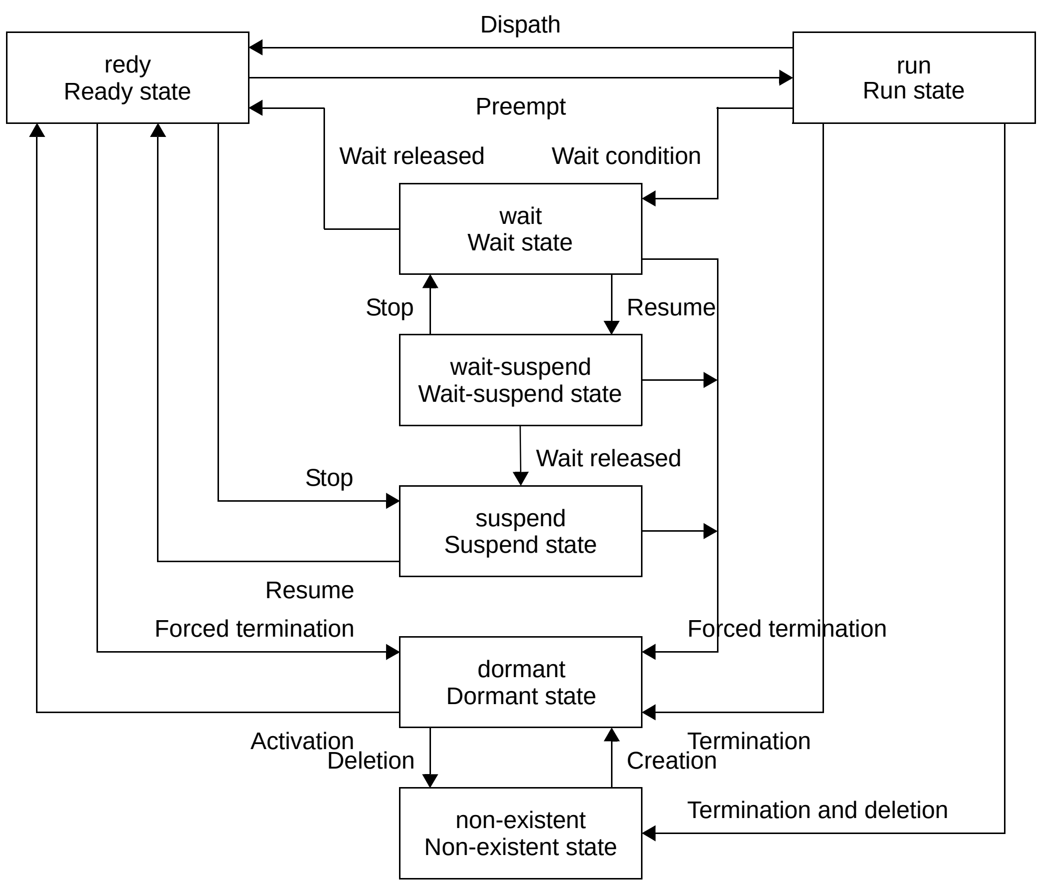
- Suspend state
  A task in this state has been forcibly stopped by another task.
  The processing of this task is resumed from the point at which it was stopped, so the values that were being used immediately before the stop are restored to the task context required for resuming the processing.

  Remark RX850 Pro supports nesting of more than one level of the suspend state (up to 127 times).

- Wait-suspend state
  This state is a combination of the wait and suspend states.
  A task in this state has entered the suspend state upon exiting from the wait state, or has entered the wait state upon exiting from the suspend state.

Task state transitions are shown in Figure 4-1.

Figure 4-1 Task State Transition



4.3 Creating Tasks

2 types of interfaces are provided in the RX850 Pro to create tasks: A task is created statically at system initialization (in the nucleus initialization module), or dynamically according to a system call issued from a processing program.

Task in the RX850 Pro consists of 3 steps: A task management area (management object) is allocated in system memory. Then, the allocated task management area is initialized. Finally, the task state is changed from the non-existent state to the dormant state.

- Static registration of a task
  To register a task statically, specify it in Task information during configuration.
  The RX850 Pro creates a task according to the information defined in the information files (system information table and system information header file) at system initialization, and makes the task manageable.

- Dynamic registration of a task
  To register a task dynamically, issue cre_tsk from a processing program (task).
  The RX850 Pro generates a task according to the information specified with parameters upon the issuance of cre_tsk, and makes the task manageable.

4.4 Activating Tasks

In task activation in the RX850 Pro, a task is switched from the dormant state to the ready state, and scheduled.
A task is activated by issuing sta_tsk, specifying the task by the parameters.

- sta_tsk
  A task specified by the parameters is switched from the dormant state to the ready state.

4.5 Terminating Tasks

In task termination in the RX850 Pro, a task is switched from the ready state, run state, wait state, suspend state, or wait-suspend state to the dormant state and excluded from the schedule by the RX850 Pro.

In the RX850 Pro, a task can be terminated in either of the following 2 ways.

Normal termination:	A task terminates upon completing all processing and when it need not be subsequently scheduled.
Forced termination:	When a number of troubles occur during processing and processing must be terminated immediately, this enables termination from another task.

The task terminates upon the issuance of the following system calls.

- ext_tsk
  The task that issued thissystem call is switched from the run state to the dormant state.

- exd_tsk
  The task that issued this system call is switched from the run state to the non-existent state.

- ter_tsk
  The task specified by the parameters is forcibly switched to the dormant state.

4.6 Deleting Tasks

In task deletion in the RX850 Pro, a task is switched from the run or dormant state to the non-existent state, and excluded from management by the RX850 Pro.

A task is deleted upon the issuance of the following system calls.

- exd_tsk
  The task that issued this system call is switched from the run state to the non-existent state.

- del_tsk
  The task specified by the parameters is switched from the dormant state to the non-existent state.

4.7 Internal Processing of Task

The RX850 Pro utilizes a unique means of scheduling to switch tasks.

Therefore, when describing a task's processing, observe the following points.

- Saving/restoring registers
  When switching tasks, the RX850 Pro saves and restores the contents of work registers in line with the function call conventions of the C compiler. This eliminates the need for coding processing to save the contents at the beginning of a task and to restore the contents at the end.
  If a task coded in assembly language uses a register for a register variable, however, the processing for saving the contents of that register must be coded at the beginning of the task, and the processing for restoring the contents at the end.

- Stack switching
  When switching tasks, the RX850 Pro switches to the special task stack of the selected task. The processing for switching the stack need not be coded at the beginning and end of the task.

- Limitations imposed on system calls
  Some of the RX850 Pro system calls cannot be issued within a task.
  The following system calls can be issued within a task:

  - Task management system calls
    cre_tsk, del_tsk, sta_tsk, ext_tsk, exd_tsk, ter_tsk, dis_dsp, ena_dsp, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid

  - Task-associated synchronization system calls
    sus_tsk, rsm_tsk, frsm_tsk, slp_tsk, tslp_tsk, wup_tsk, can_wup

  - Synchronous communication system calls
    cre_sem, del_sem, sig_sem, wai_sem, preq_sem, twai_sem, ref_sem, vget_sid, cre_flg, del_flg, set_flg, clr_flg, wai_flg, pol_flg, twai_flg, ref_flg, vget_fid, cre_mbx, del_mbx, snd_msg, rcv_msg, prcv_msg, trcv_msg, ref_mbx, vget_mid

  - Interrupt management system calls
    def_int, ena_int, dis_int, loc_cpu, unl_cpu, chg_icr, ref_icr

  - Memory pool management system calls
    cre_mpl, del_mpl, get_blk, pget_blk, tget_blk, rel_blk, ref_mpl, vget_pid

  - Time management system calls
    set_tim, get_tim, dly_tsk, def_cyc, act_cyc, ref_cyc

  - System management system calls
    get_ver, ref_sys, def_svc, viss_svc

4.7.1 Acquiring task information

Task information is acquired upon the issuance of ref_tsk.

- ref_tsk
  Task information (such as extended information or the current priority) for the task specified by the parameters is acquired.
  The contents of the task information are as follows:

  - Extended information
  - Current priority
  - Task status
  - Wait cause
  - ID number of wait object (semaphore, eventflag, etc.)
  - Number of wake-up requests
  - Number of suspend requests
  - Key ID number

4.7.2 Acquiring ID number

The ID number of a task can be acquired by issuing vget_tid.

- vget_tid
  Acquires the ID number of the task specified by the parameter.
  To manipulate a task with a system call, the ID number of the task is necessary. Whether the ID number is determined univocally by the user or automatically assigned can be specified when a task is created. If automatic assignment of the ID number is specified, however, the user cannot learn the ID number of a task. To do so, a "key ID number" is necessary. The key ID number is univocally specified when a task is created.
  By issuing this system call with this key ID number as a parameter, the ID number of the task having that key ID number can be acquired.

CHAPTER 5 SYNCHRONOUS COMMUNICATION FUNCTIONS

This chapter describes the synchronous communication functions performed by the RX850 Pro.

5.1 Outline

In an environment where multiple tasks are executed concurrently (multitasking), a result produced by a preceding task may determine the next task to be executed or affect the processing performed by the subsequent task. In other words, some task execution conditions vary with the processing performed by another task, or the processing performed by some tasks is related.

Therefore, liaison functions between tasks are required, so that task execution will be suspended to await the result output by another task or until necessary conditions have been established to enable the processing to be continued.

In the RX850 Pro, these functions are called "synchronization functions". The synchronization functions include an exclusive control function and a wait function. The RX850 Pro provides semaphores that act as the exclusive control function and eventflags that act as the wait function.

For multitasking, an inter task communication function is also required to enable one task to receive the processing result from another.

In the RX850 Pro, this function is called a "communication function". The RX850 Pro provides mailboxes that act as the communication function.

5.2 Semaphores

Multitasking requires a function to prevent the resource contention that would occur when concurrently operating multiple tasks attempt to use a limited number of resources such as an A/D converter, coprocessors, files, and programs. To implement this contention preventive function, the RX850 Pro provides non-negative counter-type semaphores.

The following system calls are used to dynamically manipulate a semaphore:

cre_sem:	Generates a semaphore.
del_sem:	Deletes a semaphore.
sig_sem:	Returns a resource.
wai_sem:	Acquires a resource.
preq_sem:	Acquires a resource (by polling).
twai_sem:	Acquires a resource (with timeout setting).
ref_sem:	Acquires semaphore information.
vget_sid:	Acquires semaphore ID number.

Remark In RX850 Pro, those elements required to execute tasks are called resources. In other words, resources comprehensively refer to hardware components such as the A/D converter and coprocessor, as well as software components such as files and programs.

5.2.1 Generating semaphores

The RX850 Pro provides 2 interfaces for generating semaphores. One enables the static generation of a semaphore during system initialization (in the nucleus initialization module). The other dynamically generates a semaphore by issuing a system call from within a processing program.

To generate a semaphore in the RX850 Pro, an area in system memory is allocated for managing that semaphore (as an object of management by the RX850 Pro), then initialized.

- Static registration of a semaphore
  To register a semaphore statically, specify it in Semaphore information during configuration.
  The RX850 Pro generates that semaphore according to the semaphore information defined in the information file (including system information tables and system information header files) during system initialization. The semaphore is subsequently managed by the RX850 Pro.

- Dynamic registration of a semaphore
  To dynamically register a semaphore, issue cre_sem from within a processing program (task).
  The RX850 Pro generates that semaphore according to the information specified with parameters when cre_sem is issued. The semaphore is subsequently managed by the RX850 Pro.

5.2.2 Deleting semaphores

A semaphore is deleted by issuing del_sem.

- del_sem
  This system call deletes the semaphore specified by the parameter.
  That semaphore is then no longer managed by the RX850 Pro.
  If a task is queued into the wait queue of the semaphore specified by this system call parameter, that task is removed from the wait queue, after which it leaves the wait state (the resource wait state) and enters the ready state.
  E_DLT is returned to the task released from the wait state as the value returned in response to the system call (wai_sem or twai_sem) that triggered the transition of the task to the wait state.

5.2.3 Returning resources

A resource is returned by issuing sig_sem.

- sig_sem
  By issuing this system call, the task returns a resource to the semaphore specified by the parameter (the semaphore counter is incremented by 0x1).
  If a task or tasks are queued into the wait queue of the semaphore specified by these system call parameter, the relevant resource is passed to the first task in the wait queue without being returned to the semaphore (thus, the semaphore counter is not incremented).
  That task is then removed from the wait queue, after which it either leaves the wait state (the resource wait state) and enters the ready state, or leaves the wait-suspend state and enters the suspend state.

5.2.4 Acquiring resources

A resource is acquired by issuing wai_sem, preq_sem, or twai_sem.

- wai_sem
  By issuing this system call, the task acquires a resource from the semaphore specified by the parameter (the semaphore counter is decremented by 0x1.)
  After issuing this system call, if the task cannot acquire the resource from the specified semaphore (no idle resource exists), the task itself is queued into the wait queue of this semaphore. Thus, the task leaves the run state and enters the wait state (the resource wait state).
  The resource wait state is canceled in the following cases, and the task returns to the ready state.

  - When sig_sem is issued.
  - When del_sem is issued and the specified semaphore is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

  Remark When a task queues in the wait queue of the specified semaphore, it is executed in the order (FIFO order or priority order) specified when that semaphore was generated (during configuration or when cre_sem was issued).

- preq_sem
  By issuing this system call, the task acquires a resource from the semaphore specified by the parameter (the semaphore counter is decremented by 0x1.)
  After this system call is issued, if the task cannot acquire the resource from the specified semaphore (no idle resource exists), E_TMOUT is returned as the return value.

- twai_sem
  By issuing this system call, the task acquires a resource from the semaphore specified by the parameter (the semaphore counter is decremented by 0x1.)
  After issuing this system call, if the task cannot acquire the resource from the specified semaphore (no idle resource

exists), the task itself is queued into the wait queue of this semaphore. Thus, the task leaves the run state and enters the wait state (the resource wait state).
The resource wait state is canceled in the following cases, and the task returns to the ready state.

- When the given wait time specified by a parameter has elapsed.
- When sig_sem is issued.
- When del_sem is issued and the specified semaphore is deleted.
- When rel_wai is issued and the wait state is forcibly canceled.

Remark When a task queues in the wait queue of the specified semaphore, it is executed in the order (FIFO order or priority order) specified when that semaphore was generated (during configuration or when cre_sem was issued).

5.2.5 Acquiring semaphore information

Semaphore information is acquired by issuing ref_sem.

- ref_sem
  By issuing this system call, the task acquires the semaphore information (extended information, queued tasks, etc.) for the semaphore specified by the parameter.
  The semaphore information consists of the following:
  - Extended information
  - Existence of waiting task
  - Current resource count
  - Maximum resource count
  - Key ID number

5.2.6 Acquiring ID number

The ID number of a semaphore can be acquired by issuing vget_sid.

- vget_sid
  Acquires the ID number of a semaphore specified by the parameter.
  To manipulate a semaphore with a system call, the ID number of the semaphore is necessary. Whether the ID number is determined univocally by the user or automatically assigned can be specified when a task is created. If automatic assignment of the ID number is specified, however, the user cannot learn the ID number of a semaphore.
  To do so, a "key ID number" is necessary. The key ID number is univocally specified when a semaphore is created.
  By issuing this system call with this key ID number as a parameter, the ID number of the semaphore having that key ID number can be acquired.

5.2.7 Exclusive control using semaphores

The following is an example of using semaphores to manipulate the tasks under exclusive control.

[Conditions]

- Task priority
  Task A > Task B

- State of tasks

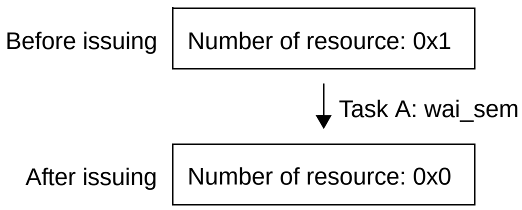
Task A:	Run state
Task B:	Ready state

- Semaphore attributes

Initial resource count:	0x1
Maximum resource count:	0x5
Task queuing method:	FIFO

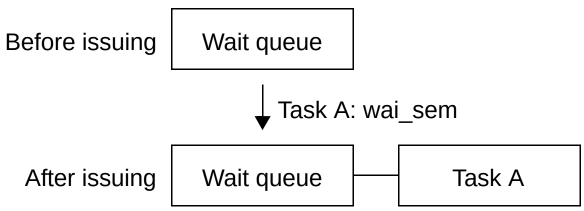
(1) Task A issues wai_sem.
The number of resources assigned to this semaphore and managed by the RX850 Pro is 0x1. Thus, the RX850 Pro decrements the semaphore counter by 0x1.
At this time, task A does not enter the wait state (the resource wait state). Instead, it remains in the run state.
The relevant semaphore counter changes as shown in Figure 5-1.

Figure 5-1 State of Semaphore Counter



(2) Task A issues wai_sem.
The number of resources assigned to this semaphore and managed by the RX850 Pro is 0x0. Thus, the RX850 Pro changes the state of task A from run to the wait state (resource wait state) and places the task at the end of the wait queue for this semaphore.
The wait queue of this semaphore changes as shown in Figure 5-2.

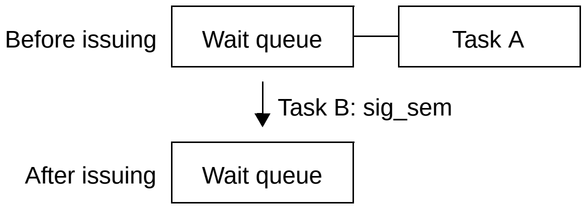
Figure 5-2 State of Wait Queue (When wai_sem Is Issued)



(3) As task A enters the resource wait state, the state of task B changes from ready to run.

(4) Task B issues sig_sem.
At this time, the state of task A that has been placed in the wait queue of this semaphore changes from the resource wait state to ready state.
The wait queue of this semaphore changes as shown in Figure 5-3.

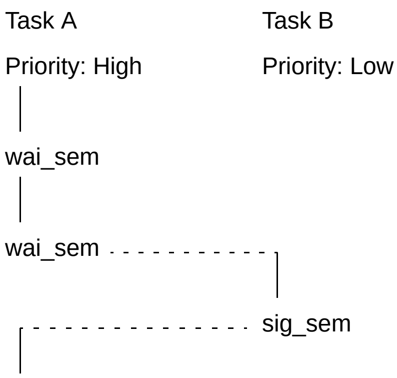
Figure 5-3 State of Wait Queue (When sig_sem Is Issued)



(5) The state of task A having the higher priority changes from ready to run.
At the same time, task B leaves the run state and enters the ready state.

Figure 5-4 shows the transition of exclusive control in steps (1) to (5).

Figure 5-4 Exclusive Control Using Semaphores



5.3 Eventflags

In multitasking, an intertask wait function, in which other tasks wait to resume execution of processing until the results of processing by a given task are output, is necessary. In such a case, it is good to have a function for other tasks to judge whether or not the "processing results output" event has occurred or not, and in the RX850 Pro, an eventflag is provided in order to realize this kind of function.

An eventflag is a set of data consisting of 1-bit flags that indicate whether a particular event has occurred. 32-bit eventflags are used in the RX850 Pro. 32 bits are handled as a set of information with each bit or a combination of bits having a specific meaning.

The following system calls regarding eventflags are used to dynamically manipulate an eventflag.

cre_flg: Generates an eventflag.
del_flg: Deletes an eventflag.
set_flg: Sets a bit pattern.
clr_flg: Clears a bit pattern.
wai_flg: Checks a bit pattern.
pol_flg: Checks a bit pattern (by polling).
twai_flg: Checks a bit pattern (with timeout setting).
ref_flg: Acquires eventflag information.
vget_fid: Acquires eventflag ID number.

5.3.1 Generating eventflags

The RX850 Pro provides 2 interfaces for generating eventflags. One is for statically generating an eventflag during system initialization (in the nucleus initialization module). The other is for dynamically generating an eventflag by issuing a system call from within a processing program.

To generate an eventflag in the RX850 Pro, an area in system memory is allocated for managing that eventflag (as an object of management by the RX850 Pro), then initialized.

- Static registration of an eventflag
  To register an event flag statically, specify it in Eventflag information during configuration.
  The RX850 Pro generates that eventflag according to the eventflag information defined in the information file (including system information tables and system information header files) during system initialization.
  Subsequently, the eventflag is managed by the RX850 Pro.

- Dynamic registration of an eventflag
  To dynamically register an eventflag, issue cre_flg from within a processing program (task).
  The RX850 Pro generates that eventflag according to the information specified by a parameter when cre_flg is issued. Subsequently, the eventflag is managed by the RX850 Pro.

5.3.2 Deleting eventflags

An eventflag is deleted by issuing del_flg.

- del_flg
  This system call deletes the eventflag specified by the parameter.
  That eventflag is then no longer managed by the RX850 Pro.
  If a task is queued into the wait queue of the eventflag specified by this system call parameter, that task is removed from the wait queue, after which it leaves the wait state (the eventflag wait state) and enters the ready state.
  E_DLT is returned to the task released from the wait state as the return value for the system call (wai_flg or twai_flg) that triggered the transition of the task to the wait state.

5.3.3 Setting a bit pattern

The eventflag bit pattern is set by issuing set_flg.

- set_flg
  This system call sets a bit pattern for the eventflag specified by the parameter.
  When this system call is issued, if the given condition for a task queued into the wait queue of the specified eventflag is satisfied, that task is removed from the wait queue.
  The task then either leaves the wait state (the eventflag wait state) and enters the ready state, or leaves the wait-suspend state and enters the suspend state.

5.3.4 Clearing a bit pattern

The eventflag bit pattern is cleared by issuing clr_flg.

- clr_flg
  This system call clears the bit pattern of the eventflag specified by the parameter.
  Note that when this system call is issued, if the bit pattern of the specified eventflag has already been cleared to zero, it is not regarded as an error.

5.3.5 Checking a bit pattern

The eventflag bit pattern is checked by issuing wai_flg, pol_flg, or twai_flg.

- wai_flg
  This system call checks whether the bit pattern is set to satisfy the wait condition required for the eventflag specified by the parameter.
  If the bit pattern does not satisfy the wait condition required this task is queued at the end of the wait queue of this eventflag. Thus, the task leaves the run state and enters the wait state (the eventflag wait state).
  The eventflag wait state is canceled in the following cases, and the task returns to the ready state.

  - When set_flg is issued and the required wait condition is set.
  - When del_flg is issued and this eventflag is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

- pol_flg
  This system call checks whether the bit pattern is set to satisfy the wait condition required for the eventflag specified by the parameter.
  If the bit pattern does not satisfy the wait condition required for the eventflag specified by this system call parameter, E_TMOUT is returned as the return value.

- twai_flg
  This system call checks whether the bit pattern is set to satisfy the wait condition required for the eventflag specified by the parameter.
  If the bit pattern does not satisfy the wait condition required for the eventflag specified by this system call parameter, the task that issues this system call is queued at the end of the wait queue for this eventflag.
  Thus, the task leaves the run state and enters the wait state (the eventflag wait state).
  The eventflag wait state is canceled in the following cases, and the task returns to the ready state.

  - Once the given wait time specified by the parameter has elapsed.
  - When set_flg is issued and the required wait condition is set.
  - When del_flg is issued and this eventflag is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

  Also, the eventflag wait conditions and processing when the conditions are established can be specified as follows in the RX850 Pro.

- Wait conditions

  - AND wait
    The wait state continues until all bits to be set to 1 in the required bit pattern have been set in the relevant eventflag.

  - OR wait
    The wait state continues until any bit to be set to 1 in the required bit pattern has been set in the relevant eventflag.

- When the condition is satisfied

  - Clearing a bit pattern
    When the wait condition specified for the eventflag is satisfied, the bit pattern for the eventflag is cleared.

5.3.6 Acquiring eventflag information

Eventflag information is acquired by issuing ref_flg.

- ref_flg
  By issuing this system call, the task acquires the eventflag information (extended information, queued tasks, etc.) for the eventflag specified by the parameter.
  Details of eventflag information are as follows:

  - Extended information
  - Existence of waiting task
  - Current bit pattern
  - Key ID number

5.3.7 Acquiring ID number

The ID number of an eventflag can be acquired by issuing vget_fid.

- vget_fid
  Acquires the ID number of the eventflag specified by the parameter.
  To manipulate an eventflag with a system call, the ID number of the eventflag is necessary. Whether the ID number is determined univocally by the user or automatically assigned can be specified when an eventflag is created.
  If automatic assignment of the ID number is specified, however, the user cannot learn the ID number of an eventflag.
  To do so, a "key ID number" is necessary. The key ID number is univocally specified when an eventflag is created.
  By issuing this system call with this key ID number as a parameter, the ID number of the eventflag having that key ID number can be acquired.

5.3.8 Wait function using eventflags

The following is an example of manipulating the tasks under wait and control using eventflags.

[Conditions]

- Task priority
  Task A > Task B

- State of tasks

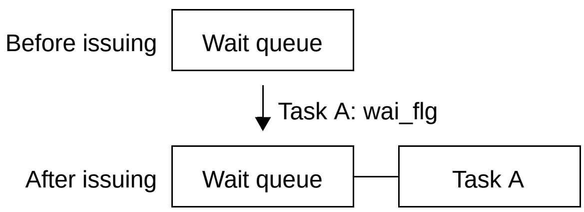
Task A:	Run state
Task B:	Ready state

- Eventflag attributes

Initial bit pattern:	0x0
Whether waiting for multiple tasks:	Disabled

(1) Task A issues wai_flg. The required bit pattern is 0x1 and the wait condition is TWF_ANDW|TWF_CLR.
The current bit pattern of the relevant eventflag managed by the RX850 Pro is 0x0. Thus, the RX850 Pro changes the state of task A from run to wait (the eventflag wait state). Task A is then queued at the end of the wait queue for this eventflag.
The wait queue of this eventflag changes as shown in Figure 5-5.

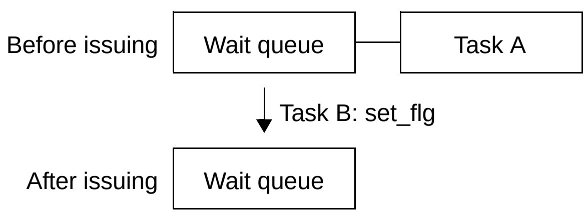
Figure 5-5 State of Wait Queue (When wai_flg Is Issued)



(2) As task A enters the eventflag wait state, the state of task B changes from ready to run.

(3) Task B issues set_flg. The bit pattern is set to 0x1.
This bit pattern satisfies the wait condition for task A that has been queued into the wait queue of the relevant eventflag. Thus, task A leaves the eventflag wait state and enters the ready state.
Since TWF_CLR was specified when task A issued the wai_flag, the bit pattern of this eventflag is cleared.
The wait queue for this eventflag changes as shown in Figure 5-6.

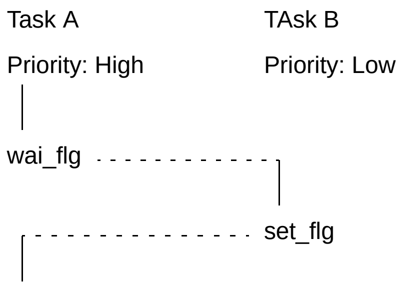
Figure 5-6 State of Wait Queue (When set_flg Is Issued)



(4) The state of task A having the higher priority changes from ready to run.
At the same time, task B leaves the run state and enters the ready state.

Figure 5-7 shows the transition of wait and control by eventflags in steps (1) to (4).

Figure 5-7 Wait and Control by Eventflags



5.4 Mailboxes

Multitasking requires an inter task communication function, so that the tasks can be informed of the results output by other tasks. To implement this function, the RX850 Pro provides mailboxes.

The mailboxes used in the RX850 Pro have 2 different queues, one dedicated to tasks and the other dedicated to messages. They can be used for both an inter task message communication function and an inter task wait function.

The following mailbox-related system calls are used to dynamically operate a mailbox.

cre_mbx: Generates a mailbox.
del_mbx: Deletes a mailbox.
snd_msg: Sends a message.
rcv_msg: Receives a message.
prcv_msg: Receives a message (by polling).
trcv_msg: Receives a message (with timeout setting).
ref_mbx: Acquires mailbox information.
vget_mid: Acquires mailbox ID number.

5.4.1 Generating mailboxes

The RX850 Pro provides 2 interfaces for generating mailboxes. One is for statically generating a mailbox during system initialization (in the nucleus initialization module). The other is for dynamically generating a mailbox by issuing a system call from within a processing program.

To generate a mailbox in the RX850 Pro, an area in system memory is allocated for managing that mailbox (as an RX850 Pro management object), then initialized.

- Static registration of a mailbox
  To register a mailbox statically, specify it in Mailbox information during configuration.
  The RX850 Pro generates the mailbox according to the mailbox information defined in the information file (including system information tables and system information header files) during system initialization.
  Subsequently, the mailbox is managed by the RX850 Pro.

- Dynamic registration of a mailbox
  To dynamically register a mailbox, issue cre_mbx from within a processing program (task).
  The RX850 Pro generates the mailbox according to the information specified by the parameter when cre_mbx is issued. Subsequently, the mailbox is managed by the RX850 Pro.

5.4.2 Deleting mailboxes

A mailbox is deleted by issuing del_mbx.

- del_mbx
  This system call deletes the mailbox specified by the parameter.
  That mailbox is then no longer managed by the RX850 Pro.
  If a task is queued into the task wait queue of the mailbox specified by this system call parameter, that task is removed from the task wait queue, after which it will leave the wait state (the message wait state) and enter the ready state.
  E_DLT is returned to the task released from the wait state as the return value for the system call (rcv_msg or trcv_msg) that triggered the transition of the task to the wait state.
  If a message is queued to the message wait queue of the specified mailbox when this system call is issued, the message is released from the wait queue and is returned to the memory pool from which the message area is acquired. Consequently, if an area other than the memory block acquired from a memory pool is used as a message area, the operation of deleting a mailbox is not guaranteed. Be sure to use a memory block acquired from a memory pool as the message area for the mailbox that may be deleted by this system call.

5.4.3 Transmitting a message

A message is transmitted from the task by issuing snd_msg.

- snd_msg
  Upon the issuance of snd_msg, the task transmits a message to the mailbox specified by the parameter.
  If a task or tasks are queued into the task wait queue of the mailbox specified by this system call parameter, the message is delivered to the first task in the task wait queue without being queued into the mailbox.
  The first task is then removed from the wait queue, after which it either leaves the wait state (the message wait state) and enters the ready state, or leaves the wait-suspend state and enters the suspend state.
  If no tasks are queued in the task wait queue of the object mailbox, the message is placed in the message wait queue of the object mailbox.

  Remark When a message queues into the message wait queue of the specified mailbox, it is executed in the order (FIFO order or priority order) specified when the mailbox was generated (during configuration or when cre_mbx was issued).

5.4.4 Receiving a message

A message is received by the task upon the issuance of rcv_msg, prcv_msg, or trcv_msg.

- rcv_msg
  Upon the issuance of this system call, the task receives a message from the mailbox specified by the parameter.
  If the task cannot receive a message from the mailbox specified by this system call parameter (no message exists in the message wait queue of that mailbox), the task that issued this system call is queued at the end of the task wait queue for this mailbox. Thus, the task leaves the run state and enters the wait state (the message wait state).
  The message wait state is canceled in the following cases and the task returns to the ready state.

  - When snd_msg is issued.
  - When del_mbx is issued and this mailbox is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

  Remark When a task queues in the task wait queue of the specified mailbox, it is executed in the order (FIFO order or priority order) specified when that mailbox was generated (during configuration or when cre_mbx was issued).

- prcv_msg
  Upon the issuance of this system call, the task receives a message from the mailbox specified by the parameter.
  If the task cannot receive a message from the mailbox specified by this system call parameter (no message exists in the message wait queue for that mailbox), E_TMOUT is returned as the return value.

- trcv_msg
  Upon the issue of this system call, the task receives a message from the mailbox specified by the parameter.
  If the task cannot receive a message from the mailbox specified by this system call parameter (no message exists in the message wait queue for that mailbox), the task that issued this system call is queued at the end of the task wait queue for this mailbox. Thus, the task leaves the run state and enters the wait state (the message wait state).
  The message wait state is canceled in the following cases and the task returns to the ready state.

  - When the given time specified by the parameter has elapsed.
  - When snd_msg is issued.
  - When del_mbx is issued and this mailbox is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

  Remark When a task queues in the task wait queue of the specified mailbox, it is executed in the order (FIFO order or priority order) specified when that mailbox was generated (during configuration or when cre_mbx was issued).

5.4.5 Messages

In the RX850 Pro, all items of information exchanged between tasks, via mailboxes, are called "messages".

Messages can be transmitted to an arbitrary task via a mailbox. In inter task communication in the RX850 Pro, however, only the start address of a message is delivered to a receiving task, enabling the task to access the message. The contents of the message are not copied to any other area.

- Allocating message areas
  NEC Electronics recommends that the memory pool managed by the RX850 Pro be allocated for messages.
  To make a memory pool area available for a message, the task should issue get_blk, pget_blk, or tget_blk.
  The first 4 bytes of each message are used as the block for linkage to the message wait queue when queued. Therefore, save messages after the first 4 bytes of the message area.

- Composition of messages
  RX850 Pro does not prescribe the length and composition of messages to be transmitted to mailboxes. The message length, except for the first 4 bytes, and its composition are defined by the tasks that communicate with each other via mailboxes.

  Remark The RX850 Pro prescribes that the first 4 bytes of each message are used as the block for linkage to the message wait queue when queued. For this reason, when a message is transmitted to the relevant mailbox, the first 4 bytes of the message must be set to 0x0 before snd_msg is issued.
  If the first 4 bytes of the message are set to a value other than 0x0 when snd_msg is issued, the RX850 Pro determines that this message has already been queued into the message wait queue. Thus, the RX850 Pro does not send the message to the mailbox and returns E_OBJ as the return value.

- Priority of messages
  The RX850 Pro can specify the priority according to which a message is to be queued. To specify the priority of a message, 2 bytes are necessary in addition to the 4 bytes of the link area that is used to queue the message to the message wait queue. Therefore, store the message in an area 6 bytes after the beginning of the message area. The message priority is specified by a positive integer of 1 to 0x7fff. The lower the value, the higher the priority.

5.4.6 Acquiring mailbox information

Mailbox information is acquired by issuing ref_mbx.

- ref_mbx
  Upon the issuance of this system call, the task acquires the mailbox information (extended information, queued tasks, etc.) for the mailbox specified by the parameter.
  The mailbox information consists of the following:

  - Extended information
  - Existence of waiting task
  - Existence of waiting message
  - Key ID number

5.4.7 Acquiring ID number

The ID number of a mailbox can be acquired by issuing vget_mid.

- vget_mid
  Acquires the ID number of a mailbox specified by the parameter.
  To manipulate a mailbox with a system call, the ID number of the mailbox is necessary. Whether the ID number is determined univocally by the user or automatically assigned can be specified when a mailbox is created. If automatic assignment of the ID number is specified, however, the user cannot learn the ID number of a mailbox. To do so, a "key ID number" is necessary. The key ID number is univocally specified when a mailbox is created.
  By issuing this system call with this key ID number as a parameter, the ID number of the mailbox having that key ID number can be acquired.

5.4.8 Inter task communication using mailboxes

The following is an example of manipulating the tasks in inter task communication using mailboxes.

[Conditions]

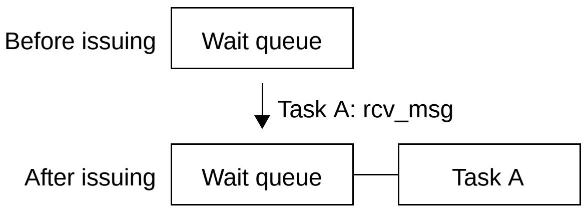
- Task priority
  Task A > Task B

- State of tasks
  Task A: Run state
  Task B: Ready state

- Mailbox attributes
  Task queuing method: FIFO
  Message queuing method: FIFO

(1) Task A issues rcv_msg.
No message is queued in the message wait queue of the relevant mailbox managed by the RX850 Pro.
Thus, the RX850 Pro changes the state of task A from run to wait (the message wait state). The task is queued at the end of the task wait queue for this mailbox.
The task wait queue for this mailbox changes as shown in Figure 5-8.

Figure 5-8 State of Task Wait Queue (When rcv_msg Is Issued)



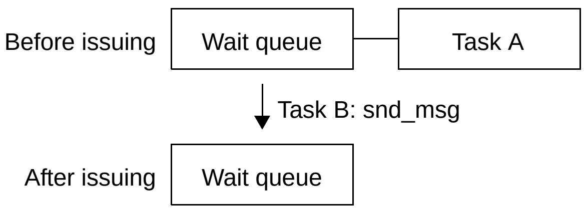
(2) As task A enters the message wait state, the state of task B changes from ready to run.

(3) Task B issues get_blk.
By means of this system call, a memory pool area is allocated for a message (as a memory block).

(4) Task B writes a message into this memory block.

(5) Task B issues snd_msg.
This changes the state of task A that has been placed in the task wait for the relevant mailbox from the message wait state to ready state.
The task wait queue for this mailbox changes as shown in Figure 5-9.

Figure 5-9 State of Task Wait Queue (When snd_msg Is Issued)

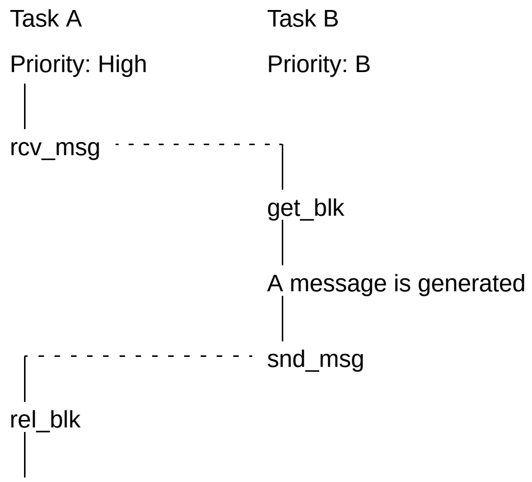


(6) The state of task A having the higher priority changes from ready to run.
At the same time, task B leaves the run state and enters the ready state.

(7) Task A issues rel_blk.
This releases the memory block allocated for the message in the memory pool.

The flow of communications between tasks as explained in (1) to (7) is shown in Figure 5-10.

Figure 5-10 Inter-Task Communication Using Mailboxes



CHAPTER 6 INTERRUPT MANAGEMENT FUNCTION

This chapter describes the interrupt management function provided by the RX850 Pro.

6.1 Outline

The RX850 Pro interrupt management function enables the following:

- Registration of an interrupt handler
- Activation of an interrupt handler
- Return from an interrupt handler
- Change or acquisition of the interrupt enable level

6.2 Interrupt Handler

The interrupt handler is a routine dedicated to interrupt servicing, which is activated immediately after an interrupt occurs. The interrupt handler is handled independently from tasks. So even if a task that has the highest priority in the system is running, the processing is suspended and control is passed to the interrupt handler.

Interrupt handlers which cannot use the functions of the RX850 Pro (such as system call issuance) in an interrupt handler (i.e. ordinary interrupt servicing) are called directly activated interrupt handlers, and the ones which can use the functions of the RX850 Pro are called indirectly activated interrupt handlers, respectively.

- Directly activated interrupt handler
  A routine dedicated to interrupt servicing that is not managed by the RX850 Pro.
  With this handler, system call execution, and multiple interrupt servicing through an indirectly activated interrupt handler are not possible. Multiple interrupt servicing through a directly activated interrupt handler is possible.
  This routine does not have any overhead for managing RX850 Pro functions, so it achieves higher response, compared with indirectly activated interrupt handlers.

- Indirectly activated interrupt handler
  A routine dedicated to interrupt servicing that is managed by the RX850 Pro.
  With this handler, system call execution, and multiple interrupt servicing through a directly activated interrupt handler and an indirectly activated interrupt handler are possible.
  This routine has overhead for managing RX850 Pro functions, so the response speed being degraded compared with directly activated interrupt handlers.

If a system call is issued while the interrupt handler is performing processing, scheduling is performed in a way specific to the RX850 Pro.

That is, if a system call (chg_pri, sig_sem, etc.) that requires task scheduling is issued during processing by the interrupt handler, the RX850 Pro merely queues the tasks into the wait queue. The actual processing of task scheduling is batched and deferred until a return from the interrupt handler has been made (by issuing the return instruction).

6.2.1 Interrupt source numbers

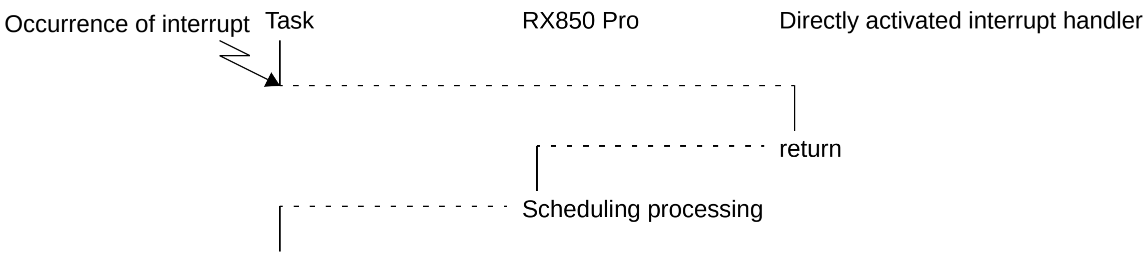
Interrupt source numbers are numbers used to indicate interrupt sources, which are calculated by "(exception code - 0x80) / 0x10", and are used to define System information clkhdr, System maximum value information maxintfactor, and Indirectly activated interrupt handler information inthdr in a system configuration file, or issuing def_int, chg_icr, or ref_icr from the processing program.

6.3 Directly Activated Interrupt Handler

A directly activated interrupt handler is a routine dedicated to interrupt processing without using the RX850 Pro upon the occurrence of an interrupt. Accordingly, a high-speed response close to the hardware limitation is expected.

The flow of the interrupt handler's operation is shown in Figure 6-1.

Figure 6-1 Flow of Processing Performed by Directly Activated Interrupt Handler



6.3.1 Registering directly activated interrupt handler

The directly activated interrupt handler is registered by allocating the handler to the handler address to which the processor transfers control if an interrupt occurs, or by setting a branch instruction that branches execution to the directly activated interrupt handler. For details, refer to "B.6 Directly Activated Interrupt Handler".

6.3.2 Processing in directly activated interrupt handler

Note the following points when coding directly activated interrupt handler processing.

- Saving and restoring the register contents
  The RX850 Pro is not involved in activation of directly activated interrupt handlers and returning from the interrupt routine. When using work registers in a directly activated interrupt handler, therefore, the code to save the work registers must be written at the beginning of the directly activated interrupt handler, and the code to restore the work registers must be written at the end of the handler.

- Switching stacks
  The stack pointer (sp) value used by a directly activated interrupt handler is the value given when an interrupt occurs.

- Restrictions on issuing system calls
  The RX850 Pro prohibits the issuance of system calls in directly activated interrupt handlers.

- Processing to return from directly activated interrupt handler
  Processing to return from a directly activated interrupt handler is performed by issuing a return instruction if the handler is written in C, or by issuing a reti instruction if the handler is written in assembly language, at the end of the handler.

  Remark The values of the global pointer (gp) and text pointer (tp) used by a directly activated interrupt handler are values given when an interrupt occurs.

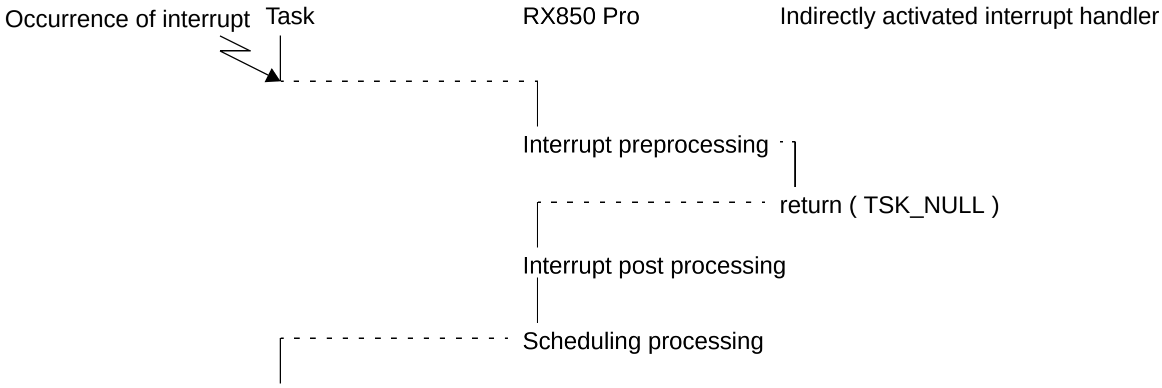
6.4 Indirectly Activated Interrupt Handler

The indirectly activated interrupt handler is an interrupt processing routine that is activated after the interrupt preprocessing of the RX850 Pro (such as saving the registers and switching the stack) has been performed if an interrupt occurs.

Because interrupt preprocessing is performed by the RX850 Pro, the indirectly activated interrupt handler has an advantage in that system calls can be issued in the handler, despite response speed being degraded compared with the directly activated interrupt handler.

Figure 6-2 shows the flow of the operation of the indirectly activated interrupt handler.

Figure 6-2 Operation Flow of Indirectly Activated Interrupt Handler



6.4.1 Registering indirectly activated interrupt handler

The RX850 Pro has 2 types of interfaces for registering an indirectly activated interrupt handler: "statically register the handler by system initialization (nucleus initialization module)" and "dynamically register the handler by issuing a system call from the processing program".

Registration of an indirectly activated interrupt handler with the RX850 Pro means allocating an area that manages the indirectly activated interrupt handler (management object) from the system memory area and initializing this area.

- Static registration
  To register an indirectly activated interrupt handler statically, specify it in Indirectly activated interrupt handler information during configuration.
  The RX850 Pro registers and manages the indirectly activated interrupt handler based on the information defined in the information files (system information table and system information header file) when system initialization is performed.

- Dynamic registration
  To dynamically register an indirectly activated interrupt handler, issue def_int from the processing program (task or non-task).
  The RX850 Pro registers and manages the indirectly activated interrupt handler based on the information specified by the parameter when def_int is issued.

Remark For the interrupt handler defined in Indirectly activated interrupt handler information and the clock handler that corresponds to the interrupt source numbers of the timer defined in System information, the configurator automatically outputs the relevant interrupt entry to the system information table, so the user does not need to write the relevant interrupt entry.
If -ne is specified as the configurator start option, however, the user must write the interrupt entry because output of the interrupt entry to the system information table is suppressed.
For details on interrupt entry, refer to "B.7 Indirectly Activated Interrupt Handler".

6.4.2 Processing in indirectly activated interrupt handler

Keep in mind the following points when describing the processing of an indirectly activated interrupt handler.

- Saving and restoring registers
  The RX850 Pro saves and restores the contents of the work registers in compliance with the function calling convention of the C compiler when it transfers control to an indirectly activated interrupt handler or when execution returns from the handler. It is therefore not necessary to save the work registers at the beginning of the indirectly activated interrupt handler and to restore the registers at the end.

- Switching stack
  The RX850 Pro switches the stack when it transfers control to an indirectly activated interrupt handler and when execution returns from the handler. It is therefore not necessary to switch the task between that for the handler and that for ordinary purposes at the beginning and end of the indirectly activated interrupt handler.
  If the stack for handler is not defined when configuration is performed, however, the stack is not switched, and the stack for ordinary purposes is used.

- Issuing system calls
  Here is a list of the system calls that can be issued in the indirectly activated interrupt handler.

  - Task management system calls
    sta_tsk, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid

  - Task-associated synchronization system calls
    sus_tsk, rsm_tsk, frsm_tsk, wup_tsk, can_wup

  - Synchronous communication system calls
    sig_sem, preq_sem, ref_sem, vget_sid, set_flg, clr_flg, pol_flg, ref_flg, vget_fid, snd_msg, prcv_msg, ref_mbx, vget_mid

  - Interrupt management system calls
    def_int, ena_int, dis_int, chg_icr, ref_icr

  - Memory pool management system calls
    pget_blk, rel_blk, ref_mpl, vget_pid

  - Time management system calls
    set_tim, get_tim, def_cyc, act_cyc, ref_cyc

  - System management system calls
    get_ver, ref_sys, def_svc, viss_svc

- Return processing from indirectly activated interrupt handler
  To exit an indirectly activated interrupt handler, issue the return instruction at the end of the handler.

  - return (TSK_NULL) instruction
    Performs return from the indirectly activated interrupt handler.

  - return (ID tskid) instruction
    Issues a wake-up request to the task specified by the parameters then returns from the indirectly activated interrupt handler.

  The RX850 Pro only manipulates the queues if a system call requiring scheduling of a task (such as chg_pri and sig_sem) is issued in an indirectly activated interrupt handler. The actual scheduling is postponed until execution returns from the indirectly activated interrupt handler, and is then performed all at once.

  Remark The return instruction does not notify an external interrupt controller of the end of processing (by issuing the EOI command). To exit from an indirectly activated interrupt handler that has been activated by an external interrupt request, therefore, notify the external interrupt controller of the end of the processing before issuing these system calls.

6.5 Disabling/Resuming Maskable Interrupt Acknowledgement

RX850 Pro provides a function for disabling or resuming the acknowledgement of maskable interrupts, so that whether maskable interrupts are acknowledged can be specified from a user processing program.

This function is used by issuing the following system calls from within a task or interrupt handler.

- loc_cpu
  This system call disables the acknowledgement of maskable interrupts, as well as the performing of dispatch processing (task scheduling).
  Once this system call has been issued, control is not passed to any other task or interrupt handler until unl_cpu is issued.

- unl_cpu
  The issue of this system call enables the acknowledgement of maskable interrupts, and resuming dispatch processing (task scheduling).
  This system call enables the acknowledgement of maskable interrupts which is disabled by loc_cpu and resumes dispatch processing.

Figure 6-3 shows the flow of control if an interrupt is not masked (normal) and Figure 6-4 shows the flow of control if loc_cpu is issued.

Figure 6-3 Control Flow if Interrupt Mask Processing Is Not Performed (Normal)

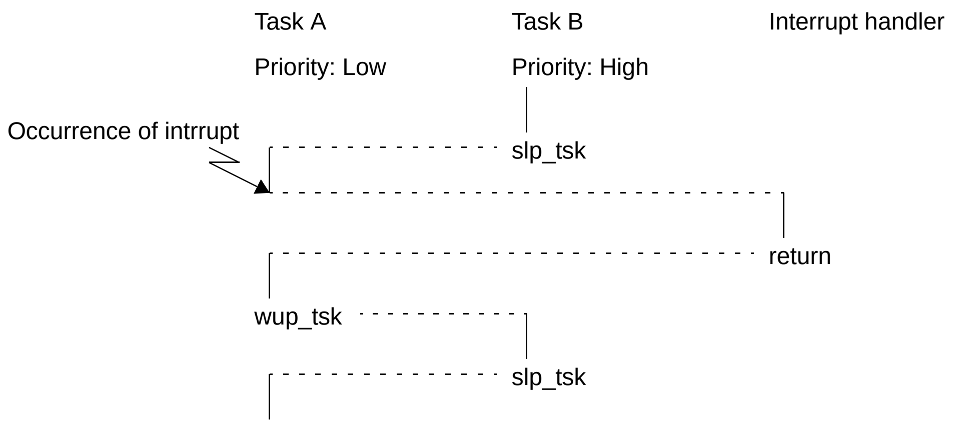
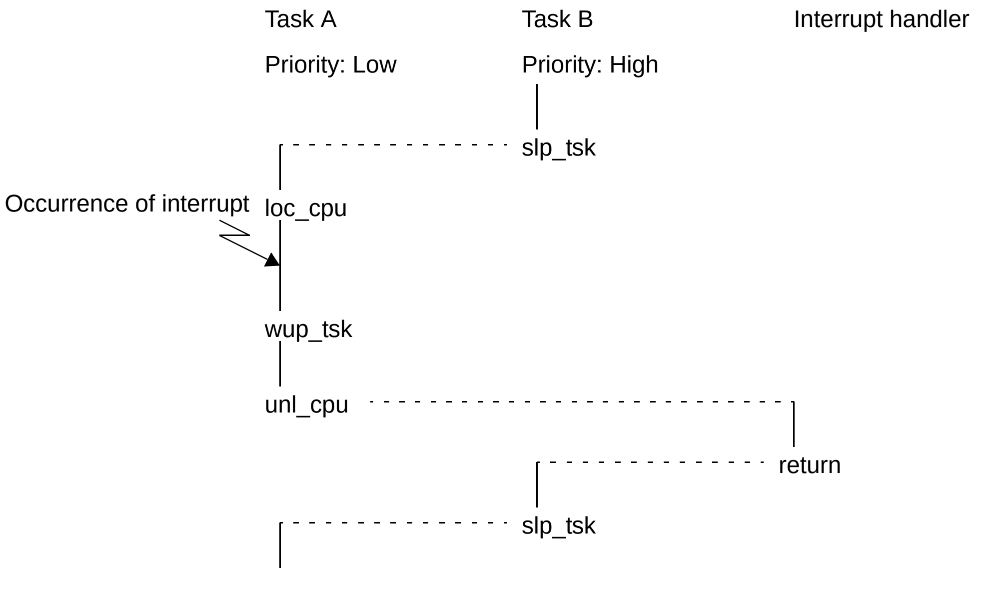


Figure 6-4 Control Flow if loc_cpu Is Issued



6.6 Changing/Acquiring Interrupt Control Register

The interrupt control register is changed or acquired by issuing chg_icr or ref_icr.

- chg_icr
  This system call changes the interrupt control register specified by the parameter.

- ref_icr
  This system call is available for acquiring the interrupt control register specified by the parameter.

Remark When the RX850 Pro is operated on the V850ES/V850E1/V850E2 core, if the interrupt control register-related chg_icr and ref_icr are issued, the desired interrupt control register may not be operated. In the RX850 Pro, the interrupt control register address is calculated from the interrupt source number. However, in the V850ES/V850E1/V850E2 core, the correct register address cannot be obtained since the alignment of the interrupt source numbers and interrupt control registers differs from other V850 microcontrollers products. Therefore, use of chg_icr and ref_icr is restricted. For manipulating the interrupt control register via an application, directly manipulate the register without using these system calls.

6.7 Non-Maskable Interrupts

A non-maskable interrupt is not subject to management based on interrupt priority and has priority over all other interrupts. It can be acknowledged even if the processor is placed in the interrupt disabled state (setting the ID flag of psw).

Therefore, even while processing is being executed by the RX850 Pro or an interrupt handler, a non-maskable interrupt can be acknowledged.

If a system call is issued during the processing of an interrupt handler that supports non-maskable interrupts, its operation cannot be assured in the RX850 Pro.

6.8 Clock Interrupts

In the RX850 Pro, time management is performed using clock interrupts, which can be generated periodically by hardware (clock controller, etc.).

If a clock interrupt is issued, RX850 Pro system clock processing is called and the processing related to time, such as the timeout wait of a task or the activation of the cyclic handler, is performed.

For details about the time management, see "CHAPTER 8 TIME MANAGEMENT FUNCTION".

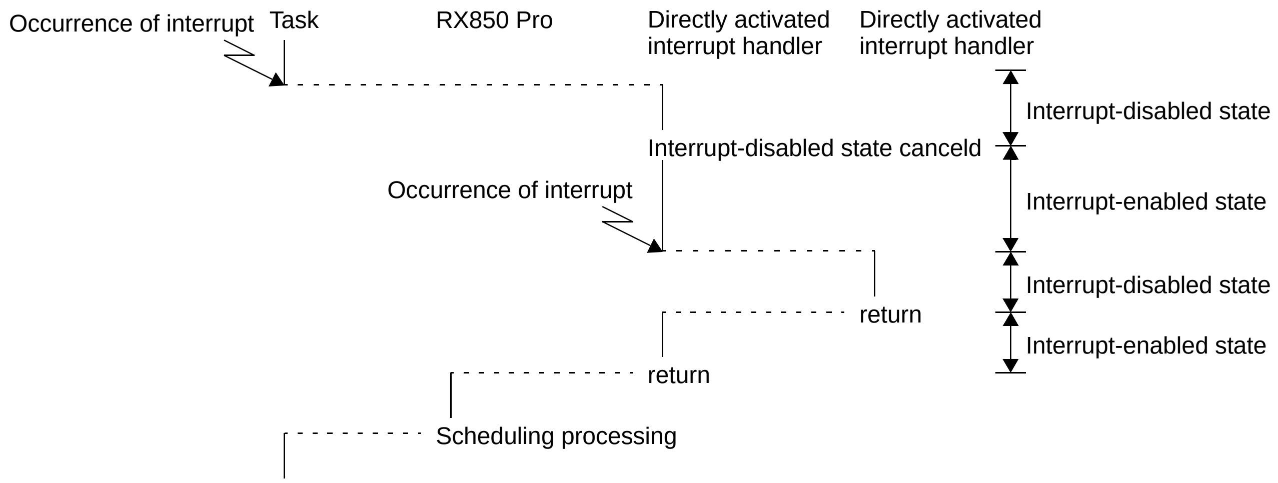
6.9 Multiple Interrupts

The occurrence of another interrupt while processing is being performed by an interrupt handler is called "multiple interrupts". The RX850 Pro also responds to multiple interrupts.

All interrupt handlers, however, start their operation in the interrupt-disabled state (setting the ID flag of psw). To enable the acknowledgement of multiple interrupts, the canceling of the interrupt disabled state should be described in the interrupt handler.

Figure 6-5 shows the flow of the processing for handling multiple interrupts.

Figure 6-5 Processing Flow for Handling Multiple Interrupts



CHAPTER 7 MEMORY POOL MANAGEMENT FUNCTION

This chapter describes the memory pool management function of the RX850 Pro.

7.1 Outline

The information table to manage systems, memory areas where the management blocks for implementing functions are allocated, and memory areas to use the memory pool are required for the RX850 Pro.
The items above are allocated in the following 4 types of memory areas.

- System Memory Pool 0 (Keyword: SPOL0)
- System Memory Pool 1 (Keyword: SPOL1)
- User Memory Pool 0 (Keyword: UPOL0)
- User Memory Pool 1 (Keyword: UPOL1)

The resource management block, task stack, interrupt handler stack, and memory for memory pool are allocated in the above memory areas. The combination of allocatable areas is as follows.

Table 7-1 Memory Information Allocation Combination

Resource Management Block	Task Stack	Interrupt Stack	Memory Pool
SPOL0	SPOL0 or SPOL1	SPOL0 or SPOL1	UPOL0 or UPOL1

The start address and size of each memory area are set using the configuration file. SPOL0 must be created.
SPOL1 needs to be created when the task stack and interrupt stack are to be allocated in other than SPOL0. UPOL0 and UPOL1 need to be created if the memory pool management function is to be used. In addition, UPOL1 can be created if UPOL0 has already been created.

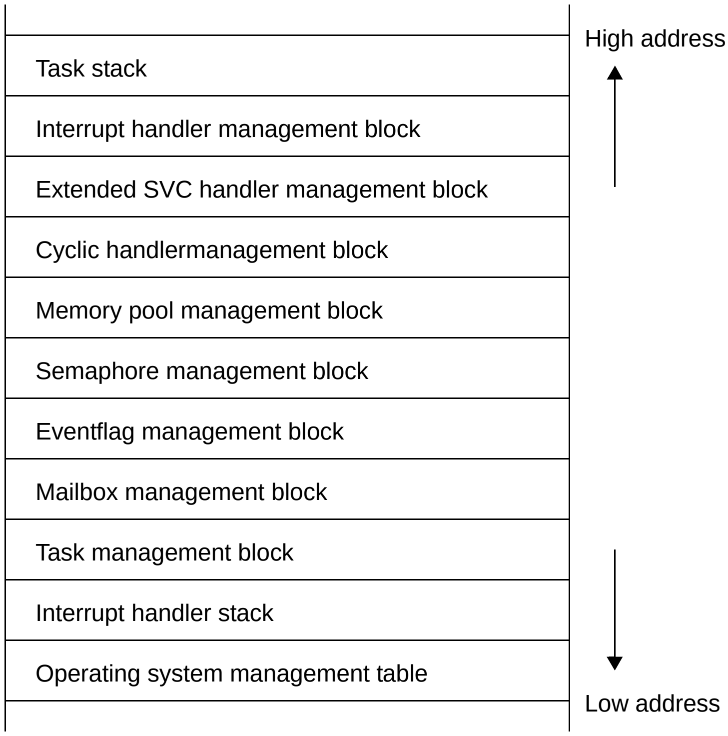
7.2 Management Objects

The objects required for implementing the functions provided by the RX850 Pro are listed below. These management objects are generated and initialized during system initialization, according to the information specified at configuration. These management objects are allocated to SPOL0 (SPOL1 also is available for task stacks and interrupt handler stacks).

- Operating system management Table
- Task management block
- Semaphore management block
- Eventflag management block
- Mailbox management block
- Memory pool management block
- Memory block management block
- Cyclic handler management block
- Extended SVC handler management block
- Memory pool
- Task stack
- Interrupt handler stack
- Interrupt handler management block

Figure 7-1 shows a typical arrangement of the management objects.

Figure 7-1 Typical Arrangement of Management Objects



7.3 Memory Pool and Memory Blocks

The RX850 Pro executes a dynamic memory pool management function through which memory areas are acquired and released during application. Using this function, the memory area is acquired if required for working, and if it becomes unnecessary, the memory area is released. This function enables efficient use of limited memory area.

Memory areas UPOL0 and UPOL1 can be used as a memory pool. Which is used (UPOL0 or UPOL1) can be specified by defining Memory pool information during configuration, or when creating a memory pool by issuing cre_mpl.

The RX850 Pro provides a variable-length memory pool, but not a fixed-length memory pool.

The memory pool consists of memory blocks and is allocated in units of memory blocks.

Dynamic generation of a memory pool and access to the memory pool are performed using the following memory pool-related system calls:

cre_mpl: Generates a memory pool.
del_mpl: Deletes the memory pool.
get_blk: Acquires a memory block.
pget_blk: Acquires a memory block (by polling).
tget_blk: Acquires a memory block (with timeout setting).
rel_blk: Release a memory block.
ref_mpl: Acquires memory pool information.
vget_pid: Acquires ID information of the memory pool.

7.3.1 Generating a memory pool

The RX850 Pro provides 2 interfaces for generating (registering) a memory pool. One enables static generation during system initialization (in the nucleus initialization module). The other enables dynamic generation by issuing a system call from within a processing program.

To generate a memory pool in the RX850 Pro, certain areas in system memory are allocated to enable management of the memory pool (as an object of RX850 Pro management) and for the memory pool main body, then initialized.

- Static registration of a memory pool
  To register a memory pool statically, specify it in Memory pool information during configuration.
  The RX850 Pro generates the memory pool, based on the information defined in the information file (including system information tables and system information header files) during system initialization. The memory pool is subsequently managed by the RX850 Pro.

- Dynamic registration of a memory pool
  To dynamically register a memory pool, issue cre_mpl from within a processing program (task).
  The RX850 Pro generates the memory pool, according to the information specified by the parameters when cre_mpl is issued. The memory pool is subsequently managed by the RX850 Pro.

Remark When a memory pool is created, the RX850 Pro uses the first 8 bytes of the memory pool as a memory pool management area, in addition to the specified size of memory. Therefore, the size of the created memory pool is "specified size + 8 bytes".

7.3.2 Deleting a memory pool

A memory pool is deleted upon the issuance of del_mpl.

- del_mpl
  This system call deletes the memory pool specified by the parameter.
  Subsequently, that memory pool is no longer subject to management by the RX850 Pro.
  If a task is queued into the wait queue of the memory pool specified by this system call parameter, that task is removed from the wait queue, leaves the wait state (the memory block wait state) and enters the ready state.
  E_DLT is returned to the task released from the wait state as the return value for the system call (get_blk or tget_blk) that triggered the transition of the task to the wait state.
  If this system call is issued, the RX850 Pro excludes the memory block managed by the specified memory pool from management. If the task has already acquired a memory block from the memory pool before this system call is issued, the operation of the memory block is not guaranteed, and care must be exercised in deleting a memory pool.

7.3.3 Acquiring a memory block

A memory block is acquired by issuing get_blk, pget_blk, or tget_blk.

Remark In the RX850 Pro, memory clear is not performed when a memory block is acquired.
Therefore, the acquired memory block's contents are undefined.

When a memory block is acquired, the RX850 Pro uses 8 bytes of the memory pool as a memory management area, in addition to the requested size of memory. The RX850 Pro also aligns the requested size by 4 bytes.

Check the remaining memory block size.

The size of the acquired memory block can be calculated by this expression:

Size of memory block (blk_siz) = align 4 (user requested size) + 8

- get_blk
  Upon the issuance of this system call, the processing program (task) acquires a memory block from the memory pool specified by the parameter.
  After the issue of this system call, if the task cannot acquire the block from the specified memory pool (because no free block of the required size exists), the task itself is queued in the wait queue of this memory pool. Thus, the task leaves the run state and enters the wait state (the memory block wait state).
  The memory block wait state is canceled in the following cases and the task returns to the ready state.

  - When rel_blk is issued and a memory block of the required size is returned.

- When del_mpl is issued and the specified memory pool is deleted.
- When rel_wai is issued and the wait state is forcibly canceled.

Remark When a task queues in the wait queue of the specified memory pool, it is executed in the order (FIFO order or priority order) specified when that memory pool was generated (during configuration or when cre_mpl was issued).

- pget_blk
  Upon the issuance of this system call, the processing program (task) acquires a memory block from the memory pool specified by a parameter.
  For this system call, if the task cannot acquire the block from the memory pool specified by this system call parameter (because no free block of the required size exists), E_TMOUT is returned as the return value.

- tget_blk
  By issuing this system call, the processing program (task) acquires a memory block from the memory pool specified by a parameter.
  After the issue of this system call, if the task cannot acquire the block from the specified memory pool (because no free block of the required size exists), the task itself is queued into the wait queue of this memory pool. Thus, the task leaves the run state and enters the wait state (the memory block wait state).
  The memory block wait state is canceled in the following cases and the task returns to the ready state.

  - When the wait time specified by the parameter has elapsed.
  - When rel_blk is issued and a memory block of the required size is returned.
  - When del_mpl is issued and the specified memory pool is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

Remark When a task queues in the wait queue of the specified memory pool, it is executed in the order (FIFO order or priority order) specified when that memory pool was generated (during configuration or when cre_mpl was issued).

7.3.4 Returning a memory block

A memory block is returned upon the issuance of rel_blk.

- rel_blk
  Upon the issuance of this system call, a processing program (task) returns a memory block to the memory pool specified by the parameter.
  For this system, if the memory block returned by this system call is of the size required by the first task in the wait queue of the specified memory pool, this block is passed to that task.
  Thus, the first task is removed from the wait queue, leaves the wait state (the memory block wait state), and enters the ready state, or leaves the wait-suspend state and enters the suspend state.

Remark1 The contents of a returned memory block area not cleared by the RX850 Pro. Thus, the contents of a memory block may be undefined when that memory block is returned.

Remark2 The RX850 Pro includes 2 different specifications for this system call.

(1) When a memory block is returned by this system call, if the first 4 bytes of the memory block are not filled with zeros, the return value E_OBJ is used for termination instead of returning the memory block.

(2) When this system call is issued, the memory block is returned even if the first 4 bytes of the memory block are not filled with zeros (return value = E_OK).

The first specification applies when the memory block is used as a mailbox's message area, and this is the specification that has been used for this system call as it has been implemented thus far in the RX850 Pro. When the memory block is used as a mailbox's message area, the first 4 bytes serve as the link area for the message's wait queue. In other words, if messages are queued in the mailbox, when this system call is issued and the memory block must be returned, in which case it is the message area linked to the queue that is returned. To prevent this, the specification requires the first 4 bytes that comprise the link area to be filled with zeros, otherwise it will be recognized as the memory block used as the message area and the return value E_OBJ will be used for termination instead of returning the memory block. Under this specification, the first 4 bytes must be cleared to zero in order to use this system call to return the memory

block.
These specifications of this system call are stored in separate libraries so that one or the other this system call specification can be used. Link to the library of this system call specification to be used.

(1) Library containing this system call that requires zero-clearing of first 4 bytes of memory block
---> librxp.a

(2) Library containing this system call that does not require zero-clearing of first 4 bytes of memory block
---> librxpm.a

Remark3 Treat a memory pool that returns a memory block the same as a memory pool specified when issuing get_blk, pget_blk, or tget_blk.

7.3.5 Acquiring memory pool information

Memory pool information is acquired by issuing ref_mpl.

- ref_mpl
  Upon the issuance of this system call, the processing program (task) acquires the memory pool information (extended information, queued tasks, etc.) for the memory pool specified by the parameter.
  The memory pool information consists of the following:

  - Extended information
  - Existence of waiting task
  - Total size of free area
  - Maximum memory block size that can acquired
  - Key ID number

7.3.6 Acquiring ID number

The ID number of a memory pool can be acquired by issuing vget_pid.

- vget_pid
  Acquires the ID number of a memory pool specified by the parameter.
  To manipulate a memory pool with a system call, the ID number of the memory pool is necessary. Whether the ID number is determined univocally by the user or automatically assigned can be specified when a memory pool is created. If automatic assignment of the ID number is specified, however, the user cannot learn the ID number of a memory pool. To do so, a "key ID number" is necessary. The key ID number is univocally specified when a memory pool is created.
  By issuing this system call with this key ID number as the parameter, the ID number of the memory pool having that key ID number can be acquired.

7.3.7 Dynamic management of memory block by memory pool

Here is an example of an operation to dynamically use the memory for tasks by using a memory pool.

[Conditions]

- Task priority
  Task A > Task B

- State of tasks

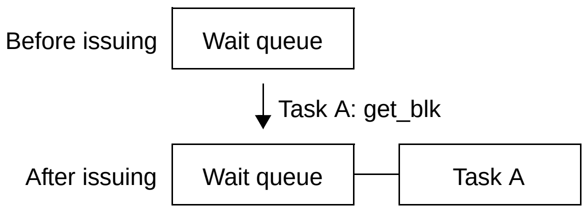
Task A:	Run state
Task B:	Ready state

- Memory pool attributes

Vacant memory block size:	0x20
Task queuing method:	FIFO

(1) Task A issues get_blk.
The requested memory block size is "0x30".
At present, the vacant memory block size of the memory pool under management of the RX850 Pro is "0x20".
Therefore, the RX850 Pro changes the state of task A from run to wait (waiting for a memory block), and queues the task to the end of the wait queue of tasks waiting for a memory pool.
At this time, this wait queue is in the state as shown in Figure 7-2.

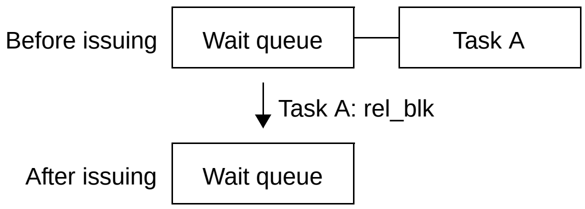
Figure 7-2 State of Wait Queue (When get_blk Is Issued)



(2) As the state of task A changes from run to wait, the state of task B changes from ready to run.

(3) Task B issues rel_blk.
The returned memory block size is "0x16".
As a result, the requested memory block size of task A queued waiting for a memory pool is satisfied and task A changes its state from wait to ready.
At this time the wait queue of tasks waiting for a memory pool is as shown in Figure 7-3.

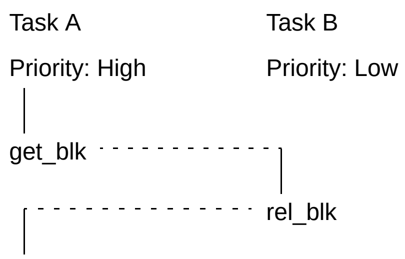
Figure 7-3 State of Wait Queue (When rel_blk Is Issued)



(4) The task A with the higher priority changes its state from ready to run.
Task B changes its state from run to ready.

Figure 7-4 shows the flow of dynamic use of memory by the memory pool explained in (1) through (4) above.

Figure 7-4 Dynamic Use of Memory by Memory Pool



CHAPTER 8 TIME MANAGEMENT FUNCTION

This chapter describes the time management function of the RX850 Pro.

8.1 Outline

Time management in the RX850 Pro is performed using clock interrupts which can be generated periodically by hardware (clock controller, etc.).

If a clock interrupt is issued, the RX850 Pro system clock processing is called and system clock update as well as processing related to time, such as delayed task wake-up, timeout, and starting of the cyclic handler, is executed.

8.2 System Clock

The system clock is a software timer that provides the time (in units of milliseconds, with a width of 48 bits) used for time management by the RX850 Pro.

The system clock is set to "0x0" during system initialization, and then updated during system clock processing, in the basic clock cycle (specified in System information during configuration) units.

Remark The system clock managed by the RX850 Pro shall have a fixed width of 48 bits. The RX850 Pro ignores any overflow (that exceeding 48 bits) for the clock value.

8.2.1 Setting and reading the system clock

The system clock setting is executed by issuing set_tim, and reading by issuing get_tim.

- set_tim
  This system call sets the time specified by the parameter to the system clock.

- get_tim
  This system call stores the current time of the system clock into the packet specified by the parameter.

8.3 Timer Operations

Real-time processing requires functions synchronized with time (timer operation functions) such as stopping the processing of a certain task for a specific time and executing the processing of a handler for specific time. The RX850 Pro therefore provides the functions of delayed wake-up of a task, timeout, and starting of a cyclic handler, as timer operation functions.

8.4 Delayed Task Wake-Up

Delayed task wake-up changes the state of a task from run to wait (the timeout wait state) and leaves the task in this state for a given period. Once this period elapses, the task is released from the wait state and returns to the ready state.

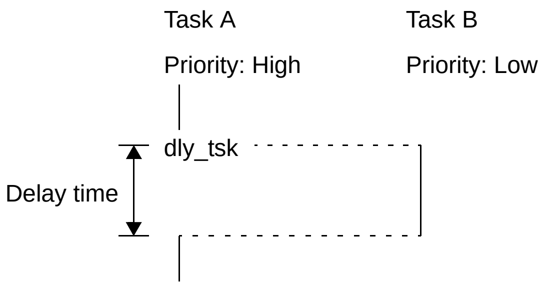
Delayed task wake-up is performed by issuing dly_tsk.

- dly_tsk
  Upon the issue of this system call, the state of the task from which this system call was issued changes from run to wait (the timeout wait state).
  The timeout wait state is canceled in the following cases and the task returns to the ready state.

  - Upon the elapse of the delay specified by a parameter.
  - Upon the issue of rel_wai and the forcible cancelation of the wait state.

Figure 8-1 shows the flow of the processing after the issue of this system call.

Figure 8-1 Flow of Processing After Issuance of dly_tsk



8.5 Timeout

If the conditions required for a certain action are not satisfied when that action is requested by a task, the timeout function changes the state of the task from run to wait (wake-up wait state, resource wait state, etc.) and leaves the task in the wait state for a given period. Once that period elapses, the timeout function releases the task from the wait state. The task then returns to the ready state.

The timeout function is enabled by issuing tslp_tsk, twai_sem, twai_flg, trcv_msg, or tget_blk.

- tslp_tsk

  Upon the issuance of this system call, one request for wake-up, issued for the task from which this system call is issued, is canceled (the wake-up request counter is decremented by 0x1).
  If the wake-up request counter of the task from which this system call is issued currently indicates 0x0, the wake-up request is not canceled (decrement of the wake-up request counter) and the task enters the wait state (the wake-up wait state) from the run state.
  The wake-up wait state is canceled in the following cases, and the task returns to the ready state.

  - When the given wait time specified by a parameter has elapsed.
  - When wup_tsk is issued.
  - When rel_wai is issued and the wait state is forcibly canceled.

- twai_sem

  Upon the issuance of this system call, the task acquires a resource from the semaphore specified by a parameter (the semaphore counter is decremented by 0x1).
  After the issuance of this system call, if the task cannot acquire a resource from the semaphore specified by the parameter (no free resource exists), the task itself is queued in the wait queue of this semaphore. Thus, the task leaves the run state and enters the wait state (the resource wait state).
  The resource wait state is canceled in the following cases, and the task returns to the ready state.

  - When the given wait time specified by a parameter has elapsed.
  - When sig_sem is issued.
  - When del_sem is issued and the specified semaphore is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

- twai_flg

  This system call checks whether the bit pattern is set so as to satisfy the wait condition required for the eventflag specified by the parameter.
  If the bit pattern does not satisfy the wait condition required for the eventflag specified by this system call parameter, the task from which this system call is issued is queued at the end of the wait queue of this eventflag. Thus, the task leaves the run state and enters the wait state (the eventflag wait state).
  The eventflag wait state is canceled in the following cases, and the task returns to the ready state.

  - When the given wait time specified by a parameter has elapsed.
  - When set_flg is issued and the required wait condition is satisfied.
  - When del_flg is issued and the specified eventflag is deleted.

- When rel_wai is issued and the wait state is forcibly canceled.

- trcv_msg
  Upon the issuance of this system call, the task receives a message from the mailbox specified by the parameter.
  After the issuance of this system call, if the task cannot receive a message from the specified mailbox (no messages exist in the message wait queue of that mailbox), the task itself is queued at the end of the task wait queue of this mailbox. Thus, the task leaves the run state and enters the wait state (the message wait state).
  The message wait state is canceled in the following cases, and the task returns to the ready state.

  - When the given time specified by a parameter has elapsed.
  - When snd_msg is issued.
  - When del_mbx is issued and this mailbox is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

- tget_blk
  Upon the issuance of this system call, the task acquires a memory block from the memory pool specified by the parameter.
  After the issuance of this system call, if the task cannot acquire the block from the specified memory pool (because no free block of the required size exists), the task itself is queued in the wait queue of this memory pool. Thus, the task leaves the run state and enters the wait state (the memory block wait state).
  The memory block wait state is canceled in the following cases, and the task returns to the ready state.

  - When the given wait time specified by a parameter has elapsed.
  - When rel_blk is issued and a memory block of the required size is returned.
  - When del_mpl is issued and the specified memory pool is deleted.
  - When rel_wai is issued and the wait state is forcibly canceled.

8.6 Cyclic Handler

The cyclic handler is an exclusive period processing routine which starts immediately when a predetermined start time arrives, and is a processing program which has optimally small overhead within the periodic processing program described by the user until execution is started.

The cyclic handler is treated as independent of the task. For this reason, even if a task with the highest priority order is being executed in the system, that processing is interrupted and the system switches to the cyclic handler's control.

The following system calls and instructions relevant to a cyclic handler are used in the dynamic operation of a cyclic handler.

def_cyc:	Registers a cyclic handler.
act_cyc:	Controls the activity state of the cyclic handler.
ref_cyc:	Acquires cyclic handler information.
return:	Performs return from the cyclic handler.

8.6.1 Registering a cyclic handler

The RX850 Pro provides 2 interfaces for registering a cyclic handler. One enables static registration during system initialization (in the nucleus initialization module). The other enables dynamic registration by issuing a system call from within a processing program.

To register a cyclic handler with the RX850 Pro, an area in system memory is allocated for managing the cyclic handler (to be managed by the RX850 Pro), then initialized.

- Static registration of a cyclic handler
  To register a cyclic handler statically, specify it in Cyclic handler information during configuration.
  The RX850 Pro performs the processing for registering the cyclic handler, based on the information defined in the information file (including system information tables and system information header files) during system initialization.
  The cyclic handler is subsequently managed by the RX850 Pro.

- Dynamic registration of a cyclic handler
  To dynamically register a cyclic handler, issue def_cyc from within a processing program (task or non-task).
  The RX850 Pro performs the processing for registering the cyclic handler, according to the information specified by the parameter when def_cyc is issued.
  The cyclic handler is subsequently managed by the RX850 Pro.

8.6.2 Activity state of cyclic handler

The activity state of a cyclic handler is used as a criterion for determining whether the RX850 Pro activated the cyclic handler.

The activity state is set when the cyclic handler is registered (during configuration or when def_cyc is issued). However, the RX850 Pro allows the user to change the activity state of the cyclic handler from a user processing program.

- act_cyc
  Upon the issuance of this system call, the activity state of the cyclic handler is switched ON/OFF, as specified by the parameter.

TCY_OFF:	Switches the activity state of the cyclic handler to OFF.
TCY_ON:	Switches the activity state of the cyclic handler to ON.
TCY_INI:	Initializes the cycle counter of the cyclic handler.

While the RX850 Pro is running, the cycle counter continues to count even when the activity state of the cyclic handler is OFF. In some cases, when act_cyc is issued to switch the activity state of the cyclic handler from OFF to ON, the first restart request could be issued sooner than the activation time interval specified when it was registered (during configuration or upon the issuance of def_cyc). To prevent this, the user must specify TCY_INI to initialize the cycle counter as well as TCY_ON to restart the cyclic handler when issuing act_cyc. The first restart request will then be issued in sync with the time interval, specified when it was registered.

Figure 8-2 and Figure 8-3 show the flow of processing after the issuance of act_cyc from a processing program to switch the activity state of the cyclic handler from OFF to ON.

In those figures, ΔT indicates the activation time interval specified upon the registration of the cyclic handler. In addition, the relationship between Δt and ΔT in Figure 8-2 is assumed to be $\Delta t \leq \Delta T$.

Figure 8-2 Flow of Processing After Issuance of act_cyc (TCY_ON)

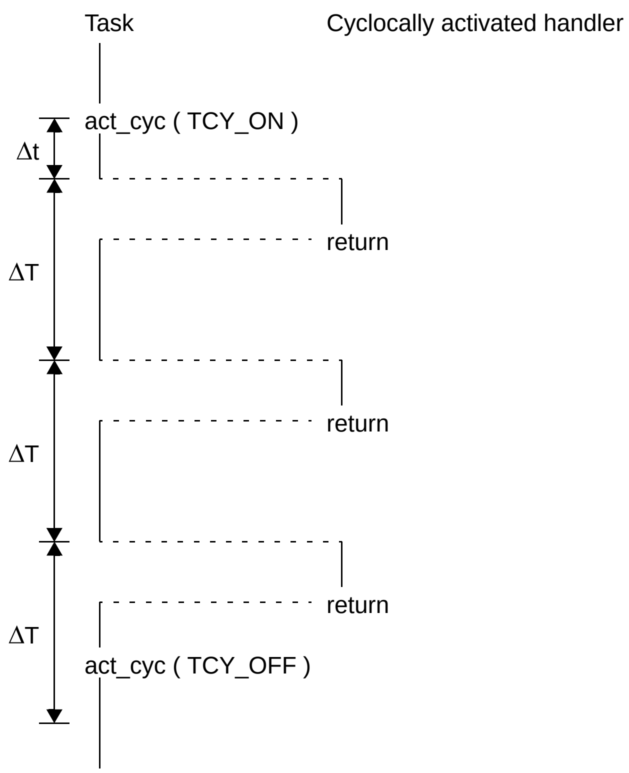
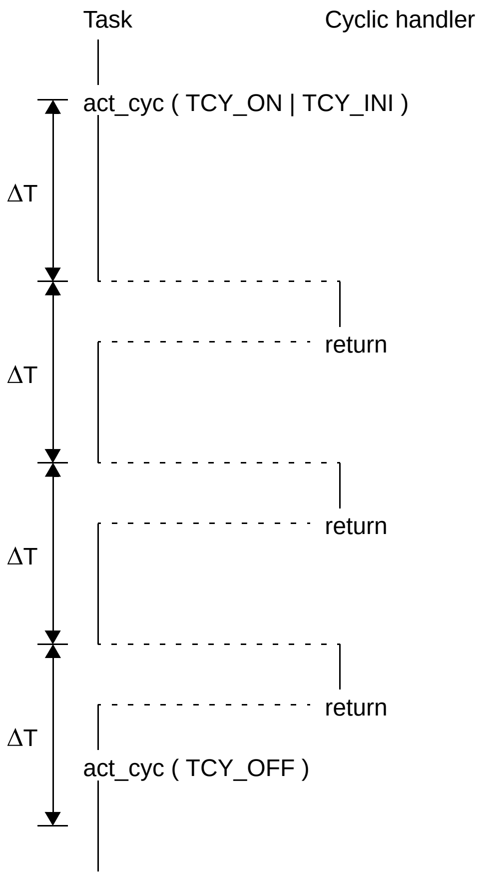


Figure 8-3 Flow of Processing After Issuance of act_cyc (TCY_ON|TCY_INI)



8.6.3 Internal processing performed by cyclic handler

After the occurrence of a clock interrupt, the RX850 Pro performs preprocessing for interruption before control is passed to the cyclic handler. When control is returned from the cyclic handler, the RX850 Pro performs interrupt postprocessing.

When describing the processing to be performed by the activated interrupt handler, note the following:

- Saving/restoring the registers
  Based on the function call protocol for the C compiler, the RX850 Pro saves the work registers when control is passed to the cyclic handler, and restores them upon the return of control from the handler. Therefore, the cyclic handler does not have to save the work registers when it starts, nor restore them upon the completion of its processing. Save/restoration of the registers should not be coded in the description of the cyclic handler.

- Stack switching
  The RX850 Pro performs stack switching when control is passed to the cyclic handler and upon a return from the handler. Therefore, the cyclic handler does not have to switch to the interrupt handler stack when it starts, nor switch to the original stack upon the completion of its processing. However, if the interrupt handler stack is not defined during configuration, stack switching is not performed and system continues to use that stack being used upon the occurrence of an interrupt.

- Limitations imposed on system calls
  The following lists the system calls that can be issued during the processing performed by a cyclic handler:

  - Task management system calls
    sta_tsk, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid

  - Task-associated synchronization system calls
    sus_tsk, rsm_tsk, frsm_tsk, wup_tsk, can_wup

  - Synchronous communication system calls
    sig_sem, preq_sem, ref_sem, vget_sid, set_flg, clr_flg, pol_flg, ref_flg, vget_fid, snd_msg, prcv_msg, ref_mbx, vget_mid

  - Interrupt management system calls
    def_int, ena_int, dis_int, chg_icr, ref_icr

  - Memory pool management system calls
    pget_blk, rel_blk, ref_mpl, vget_pid

  - Time management system calls
    set_tim, get_tim, def_cyc, act_cyc, ref_cyc

  - System management system calls
    get_ver, ref_sys, def_svc, viss_svc

- Return processing from the cyclic handler
  Return processing from the cyclic handler is performed by issuing a return instruction upon the completion of the processing performed by cyclic handler.
  When a system call (chg_pri, sig_sem, etc.) that requires task scheduling is issued during the processing of a cyclic handler, RX850 Pro merely queues that task into the wait queue. The actual task scheduling is batched and deferred until return from the cyclic handler has been completed (by issuing a return instruction).

8.6.4 Acquiring cyclic handler information

Information related to a cyclic handler is acquired by issuing ref_cyc.

- ref_cyc
  By issuing this system call, the task acquires information (including extended information, remaining time, etc.) related to the cyclic handler specified by a parameter.
  The cyclic handler information consists of the following:

  - Extended information
  - Time remaining until the next start of the cyclic handler
  - Current activity state

8.6.5 Interrupts in cyclic handler

Interrupts are disabled for the cyclic handler at startup. To use interrupts during cyclic handler processing, enable interrupts at the beginning of the handler.

Since the RX850 Pro provides 2 types of nucleus common parts (rxtncore.o and rxcore.o), the interrupts that can be acknowledged within the cyclic handler differ depending on the nucleus common part used.

- When rxtmcore.o is used
  Although the cyclic handler is called from the clock handler, only the interrupts with a higher priority than clock interrupts can be acknowledged because the interrupt processing is not performed during clock handler execution. In addition, since clock interrupts are held pending even when interrupts are enabled, to execute a time-consuming processing within the cyclic handler, caution is required because displacement may occur between the time that has actually elapsed and the time managed by the RX850 Pro.
  Because the cyclic handler is developed as an indirectly activated interrupt handler, it operates on the handler stack at execution.

- When rxcore.o is used
  Although the cyclic handler is called from the clock handler, all levels of interrupts can be acknowledged because the interrupt processing is performed during clock handler execution.

Because the cyclic handler is developed as an indirectly activated interrupt handler, it operates on the handler stack at execution.

CHAPTER 9 SCHEDULER

This chapter explains the task scheduling performed by the RX850 Pro.

9.1 Outline

By monitoring the dynamically changing task states, the RX850 Pro scheduler manages and determines the sequence in which tasks are executed, and assigns a processing time to a specific task.

9.2 Drive Method

The RX850 Pro scheduler uses an event-driven technique, in which the scheduler operates in response to the occurrence of some event.

The "occurrence of some event" means the issue of a system call that may cause a task state change, the issue of a return instruction that causes a return from a handler, or the occurrence of a clock interrupt.

When these phenomena occur, task scheduling processing is executed with the scheduler driving.

The following system calls can be used to drive the scheduler.

- Task management system calls
  sta_tsk, ext_tsk, exd_tsk, ena_dsp, chg_pri, rot_rdq, rel_wai

- Task-associated synchronization system calls
  rsm_tsk, frsm_tsk, slp_tsk, tslp_tsk, wup_tsk

- Synchronous communication system calls
  del_sem, sig_sem, wai_sem, twai_sem, del_flg, set_flg, wai_flg, twai_flg, del_mbx, snd_msg, rcv_msg, trcv_msg

- Interrupt management system calls
  unl_cpu

- Memory pool management system calls
  del_mpl, get_blk, tget_blk, rel_blk

- Time management system call
  dly_tsk

9.3 Scheduling Method

The RX850 Pro uses the priority and FCFS (First-Come, First-Served) scheduling method. When driven, the scheduler checks the priority of each task that can be executed (in the run or ready state), selects the optimum task, and assigns a processing time to the selected task.

9.3.1 Priority method

Each task is assigned a priority that determines the sequence in which it will be executed.

The scheduler checks the priority of each task that can be executed (in the run or ready state), selects the task having the highest priority, and assigns a processing time to the selected task.

Remark In the RX850 Pro, a task to which a smaller value is assigned as the priority level has a higher priority.

9.3.2 FCFS method

The RX850 Pro can assign the same priority to more than one task. Because the priority method is used for task scheduling, there is the possibility of more than one task having the highest priority being selected.

Among those tasks having the highest priority, the scheduler selects the first to become executable (the task that has been in the ready state for the longest time) and assigns a processing time to the selected task.

9.4 Implementing a Round-Robin Method

In scheduling based on the priority and FCFS methods, even if a task has the same priority as that currently running, it cannot be executed unless the task to which a processing time has been assigned first enters another state or relinquishes control of the processor.

The RX850 Pro provides system calls such as rot_rdq to implement a scheduling method (round-robin method) that can overcome the problems incurred by the priority and FCFS methods.

The round-robin method can be implemented as follows:

[Conditions]

- Task priority

  Task A = Task B = Task C

- State of tasks

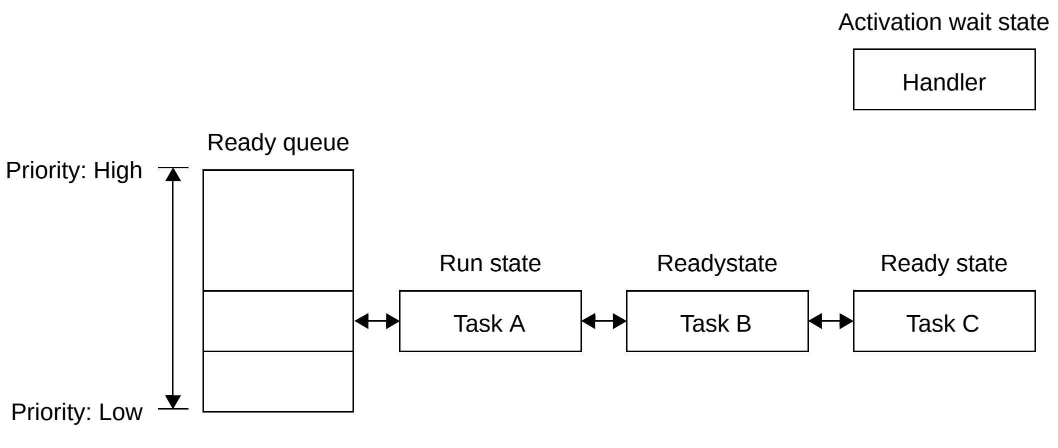
Task A:	Run state
Task B:	Ready state
Task C:	Ready state

- Cyclic handler attributes

Activity state:	ON
Activation interval:	ΔT (unit: Basic clock cycle)
Processing:	Rotation of the ready queues (issue of rot_rdq)

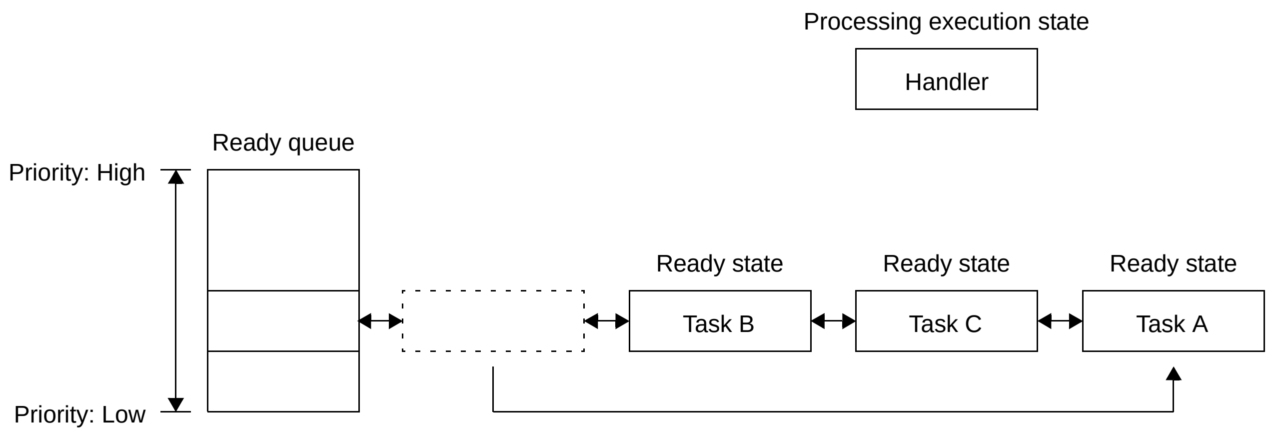
(1) Task A is currently running.
The other tasks (B and C) have the same priority as task A, but they cannot be executed unless task A enters another state or relinquishes control of the processor.
The ready queue becomes as shown in Figure 9-1.

Figure 9-1 Ready Queue State (1)



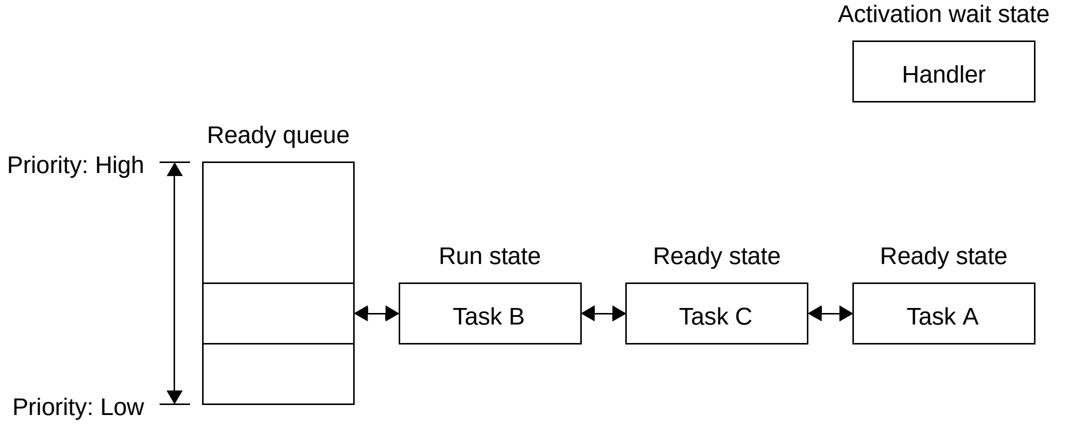
(2) Cyclic handler starts when the predetermined period of time has passed and issues rot_rdq.
In this way, task A is queued at the tail end of the ready queue in accordance with its priority level and changes from the run state to ready state.
The ready queue changes to the state shown in Figure 9-2.

Figure 9-2 Ready Queue State (2)



(3) Task A changes from the run state to the ready state and task B changes from the ready state to the run state.
Figure 9-3 shows the ready queue state at this time.

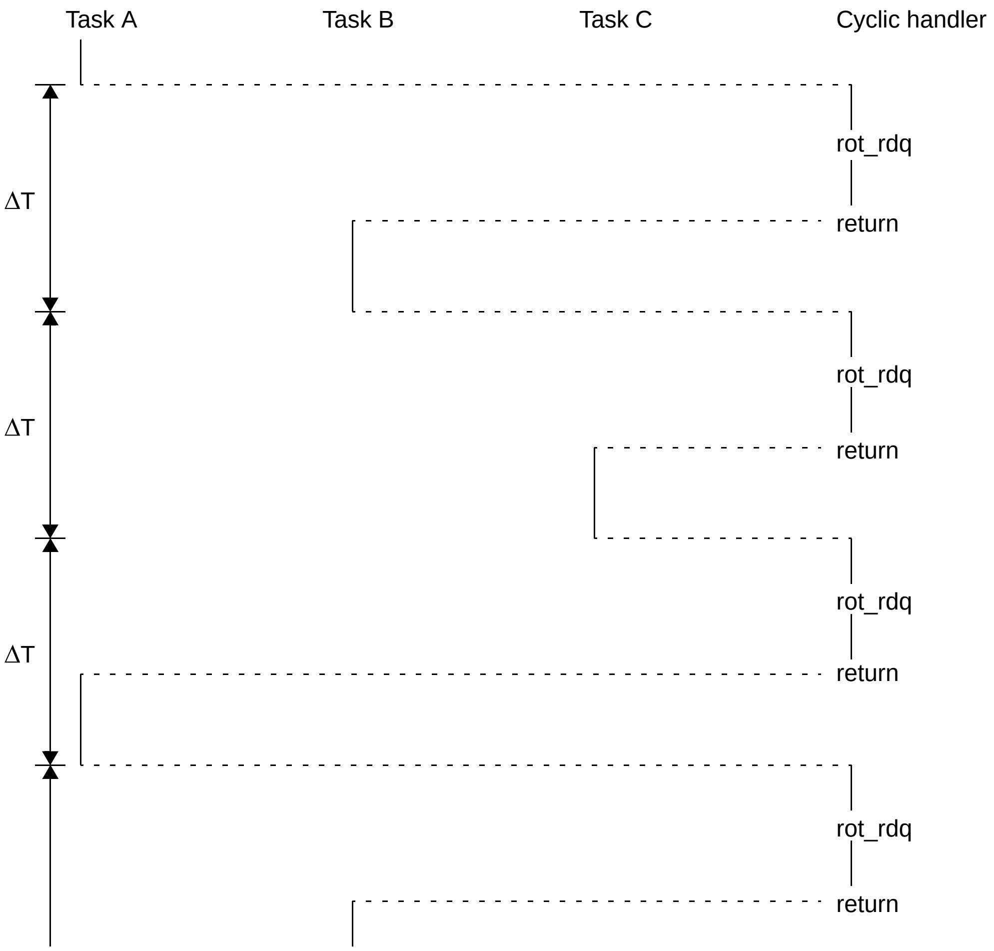
Figure 9-3 Ready Queue State (3)



(4) By issuing rot_rdq from the cyclic handler, which is started at constant intervals, the scheduling method (round-robin method) in which tasks are switched every time the specified period (ΔT) elapses is implemented.

Figure 9-4 shows the processing flow when the round-robin method is used.

Figure 9-4 Flow of Processing by Using Round-Robin Method



9.5 Scheduling Lock Function

In the RX850 Pro a function is offered which drives the scheduler from a user processing program (task) and which disables or resumes dispatch processing (task scheduling processing).

This function is implemented by issuing the following system calls from within a task.

- dis_dsp
  Disables dispatching (task scheduling).
  If this system call is issued, control is not passed to another task until ena_dsp is issued.

- ena_dsp
  Resumes dispatching (task scheduling).
  When dis_dsp has been issued, if a system call that requires task scheduling (such as chg_pri or sig_sem) is issued, the RX850 Pro merely executes processing such as wait queue operation until this system call is issued. Actual scheduling is delayed and batch-executed upon the issuance of this system call.

- loc_cpu
  Disables the acknowledgement of maskable interrupts, then disables dispatching (task scheduling).
  If this system call is issued, control will not be passed to another task or handler until unl_cpu is issued.

- unl_cpu
  Enables the acknowledgement of maskable interrupts, then restarts dispatching (task scheduling).
  If a maskable interrupt has occurred between the issuance of loc_cpu and that of this system call, transfer of control to the corresponding interrupt handling (processing of the interrupt handler) is delayed until this system call is issued. Also, if a system call which is necessary for task scheduling processing (such as chg_pri or sig_sem) is issued during the interval after loc_cpu is issued and until this system call is issued, only processing of wait queue operations is delayed until this system call is issued, being performed by batch processing.

The flow of control if scheduling processing is not delayed (normal) is shown in Figure 9-5 and the flow of control if dis_dsp and loc_cpu are issued is shown in Figure 9-6 and Figure 9-7.

Figure 9-5 Flow of Control if Scheduling Processing Is Not Delayed (Normal)

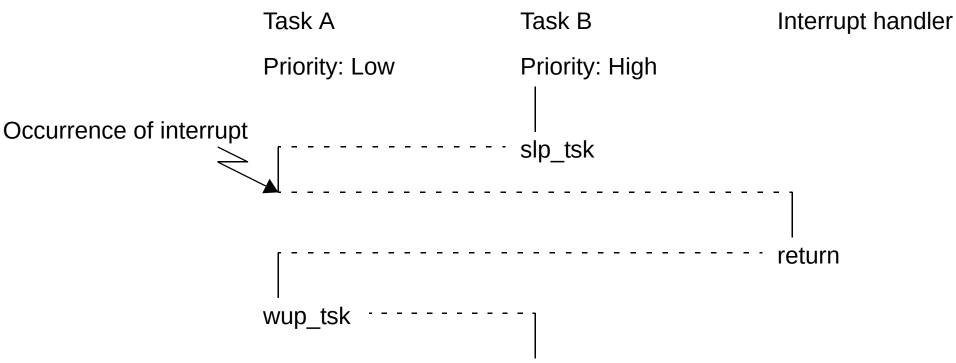


Figure 9-6 Flow of Control if dis_dsp Is Issued

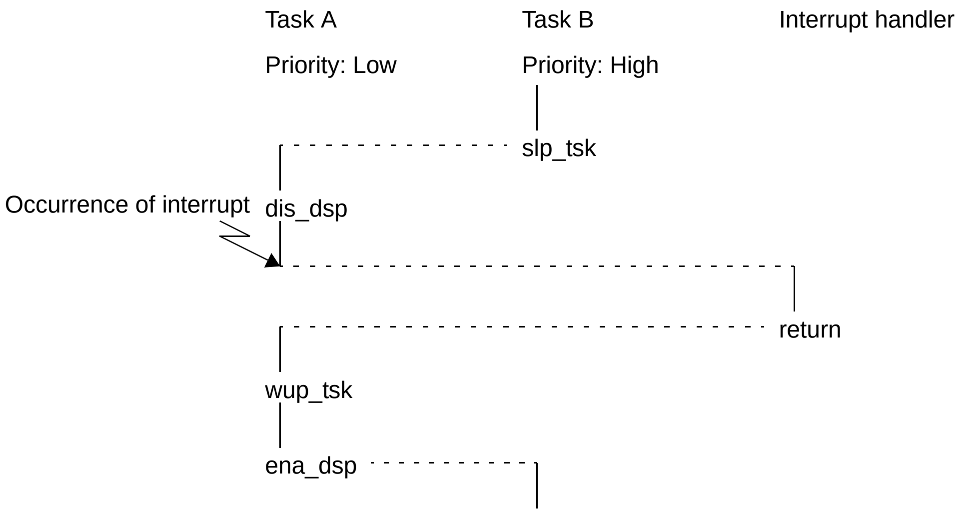
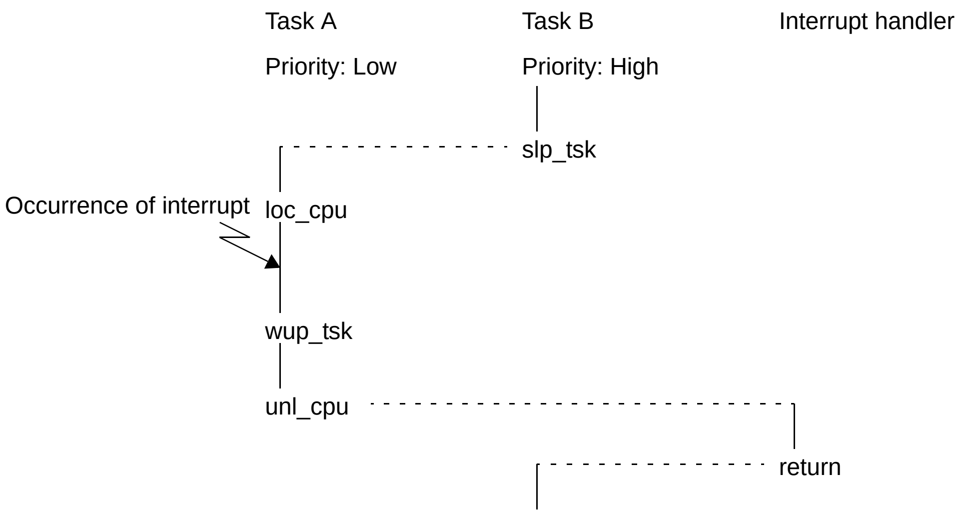


Figure 9-7 Flow of Control if loc_cpu Is Issued



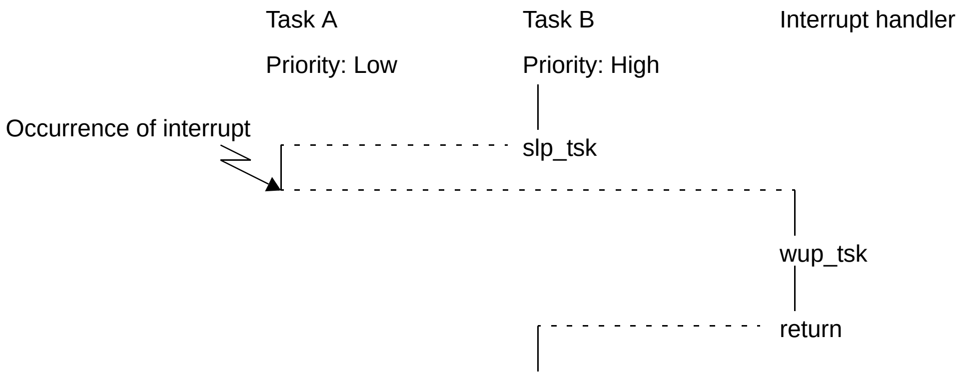
9.6 Scheduling While Handler Is Operating

To quickly terminate handlers (interrupt handlers and cyclic handlers), the RX850 Pro delays the driving of the scheduler until processing within the handler terminates.

Therefore, if a system call (such as chg_pri and sig_sem) that requires task scheduling during handler processing is issued, the RX850 Pro only performs processing such as the wait queue operation, but the actual scheduling is performed all at once after processing to return from the handler (by executing a return instruction or the like) is completed.

Figure 9-8 shows the control flow when a handler issues a system call that requires scheduling.

Figure 9-8 Flow of Control if wup_tsk Is Issued



9.7 Idle Handler

The idle handler is started from the scheduler if all the tasks (user defined tasks) are not in the run state or not in the ready state, that is, if there is not even one task which is an object of RX850 Pro scheduling in the system.

The processing of the idle handler is to switch the CPU to the HALT state. Therefore, if there is not even one task in the system, the RX850 Pro switches the CPU to the HALT state.

However, this idle handler cannot switch the CPU to the IDLE or STOP state. To switch to the IDLE or STOP state, or to describe idle processing, create a task with the lowest priority and use it as an idle task. This realizes processing identical to the idle handler. However, since the HALT, IDLE, or STOP state is released by an interrupt, be sure not to leave interrupts in a disabled state in the idle task.

CHAPTER 10 SYSTEM INITIALIZATION

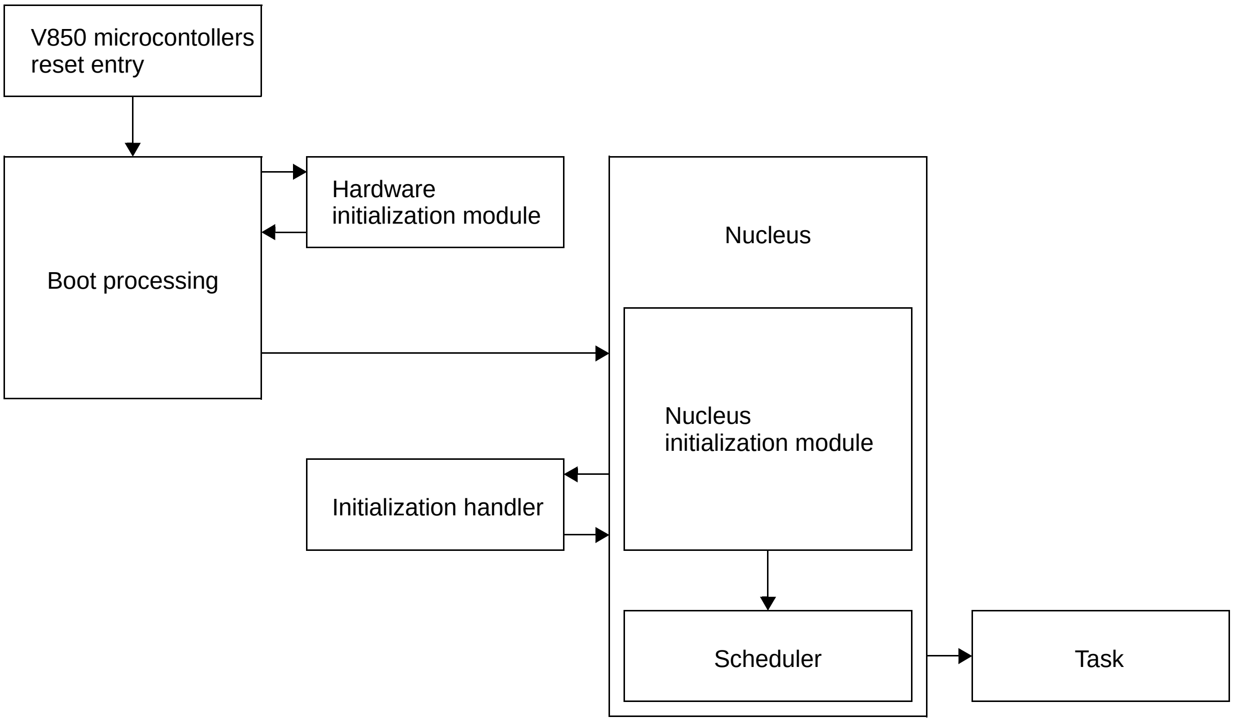
This chapter explains the system initialization performed by the RX850 Pro.

10.1 Outline

System initialization consists of initializing the hardware required by the RX850 Pro, as well as initializing the software. In other words, in the RX850 Pro, the processing performed immediately after the system has been started is system initialization.

Figure 10-1 shows the flow of system initialization.

Figure 10-1 Flow of System Initialization



10.2 Boot Processing

Boot processing is the function assigned to the V850 microcontrollers reset entry (handler address: 0x0) and the first function executed in system initialization. The files boot.s are used in the sample boot processing (function name: __boot).

The following must be performed as part of the boot processing.

- Setting of stack pointer (sp) used in boot processing
- Setting of text pointer (tp) and global pointer (gp)
- Issuance of jarl instruction to transfer control to hardware initialization module
- Setting of symbol _sit to r10 register address
- Setting of symbol __rx_start to lp register
- Issuance of jmp instruction to transfer control to nucleus initialization module

In the sample boot processing, the processing can be rewritten to adapt to user needs.

10.3 Hardware Initialization Module

The hardware initialization module is a function called from the boot processing and it is prepared for initializing the hardware in the execution environment (target system). The file init.c is used in the sample initialization (function name: reset).

In this hardware initialization module, the following processing is performed.

- Initialization of the internal unit
    - Initialization of an interrupt controller
    - Initialization of a clock controller
- Initialization of a peripheral controller
- Returns control to boot processing

The hardware initialization module depends on the hardware configuration of the execution environment.

Designing this section into the LSI improves portability to the target system and simplifies customization. Rewrite in accordance with the user execution environment.

10.4 Nucleus Initialization Module

The nucleus initialization module is a function called after the boot processing completion and it generates and initializes the management objects based on the information (such as task information or semaphore information) described in the information files (system information table and system information header file). The RX850 Pro is activated after completion of this processing. This processing section is included in the nucleus library.

The nucleus initialization module performs the following processing.

- Generation/initialization of management objects
    - Task generation
    - Generation/initialization of a semaphore
    - Generation/initialization of eventflags
    - Generation/initialization of a memory pool
    - Registration of the indirectly activated interrupt handler
    - Registration of the cyclic handler
    - Registration of the extended SVC handler
- Activation of an initial task
- Activation of the system task (idle task)
- Calling of the initialization handler
- Transfer of control to the scheduler

10.5 Initialization Handler

The initialization handler is a function called from the nucleus initialization module and used if some processing is to be executed before the activation of the RX850 Pro. The file varfunc.c is used in the sample processing (function name: varfunc).

The initialization handler performs the following processing.

- Copying of an initialization data
- Returns control to the nucleus initialization module

Remark1 When passing control from the nucleus initialization module to the initialization handler, the RX850 Pro switches the current stack to the system stack that is specified in System information during configuration.

Remark2 When passing control from the nucleus initialization module to the initialization handler, the RX850 Pro switches the values of the text pointer (tp) and global pointer (gp) to values that are defined in Initialization handler information during configuration.

Remark3 The RX850 Pro performs no operations on the element pointer (ep). The ep value used during the initialization handler processing therefore differs from the value set during boot processing.

Remark4 The initialization handler is called before the RX850 Pro completes all of the initialization processing. Therefore, if interrupts for the initialization handler are enabled or a system call is issued by the initialization handler, the operation is not guaranteed.

10.6 Interrupt Entry

An interrupt entry is an instruction that is executed if an interrupt occurs, and is assigned to the "interrupt handler address" of the V850 microcontrollers. The interrupt entry must be defined for all the interrupts used by the user, and must be described in assembly language. The files entry.s are used in the sample interrupt entry.

CHAPTER 11 INTERFACE LIBRARY

This chapter describes the interface libraries provided by the RX850 Pro.

11.1 Outline

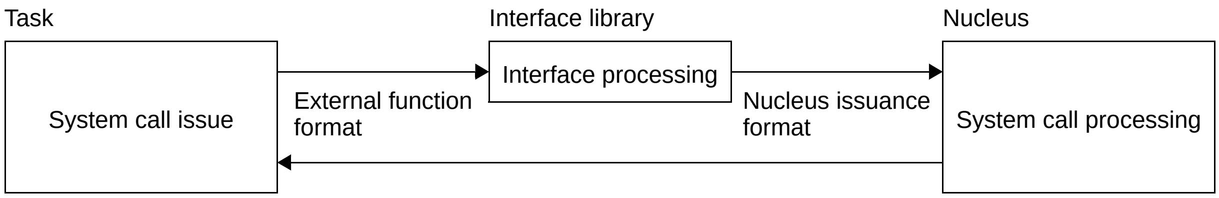
The RX850 Pro provides an interface library that is positioned between the user's processing program and the nucleus of the RX850 Pro. The interface library performs the data setting and other processing required for the nucleus to carry out its functions, before passing control to the nucleus.

Processing programs (tasks, non-tasks) coded in C are generated in the external function format which is used to issue system calls and call extended SVC handlers. The issuance format that can be recognized by the nucleus (nucleus issuance format), however, differs from the external function format.

This necessitates a procedure (interface) for converting the external function format, used to issue system calls and to call extended SVC handlers, into a format that can be issued to the nucleus. This type of interface between a processing program and the nucleus is provided for each system call. The interface library contains a collection of such interfaces.

Figure 11-1 shows the interface library position.

Figure 11-1 Position of Interface Library



11.2 Processing in the Interface Library

The interface library performs the following processing.

- Sets necessary information to table managed by nucleus
- Sets necessary data to registers
- Sets error values of system calls (except errors set in nucleus) and returns control to processing program

By preparing the interface library, the nucleus and the processing program of the user can be easily separated. The interface library is linked to the user application. Even if it is necessary to change the processing program of the user after the entity of the nucleus has been stored in ROM, therefore, the ROM storing the nucleus entity does not have to be changed. Also, the user can create load modules while separating them into different sections. The syntax of the interface library for calling system calls is described in "11.5 System Call Interface Library", and the syntax of the interface library of extended SVC handler is described in "11.6 Extended SVC Handler Interface Library". Refer to these sections for details.

11.3 Types of Interface Libraries

The RX850 Pro supplies 2 types of interface libraries: one that has a function to check the parameters of system calls and another that does not. Specify which of the interfaces is to be embedded in the system during linkage.

If the library with the parameter check function is used, and if an illegal parameter is specified when a system call is issued, a return value is always returned. If the library without the parameter check function is used, no return value is returned even if an illegal parameter is specified when a system call is issued.

These 2 types of libraries are used for different applications. For example, the library with the parameter check function may be used for debugging, while the library without the parameter check function is actually embedded in the system, in order to improve the cost effectiveness of the program and save the memory capacity.

Remark1 Errors in which return values are returned with the library which does not have a parameter check function are marked by "*" in the system call return value column in "CHAPTER 12 SYSTEM CALLS".

Remark2 When the library without the parameter check function is used, if errors occur in which return values are not returned, the operation of the application system cannot be guaranteed.

11.4 Change Interface Libraries

To change the interface library as necessary, it is necessary to rewrite the interface library.

Once an interface library has been modified, it must be assembled and then defined again as a library.

11.5 System Call Interface Library

The main operation of the system call interface library is detailed below.
The method of system call parameter passing, however, complies with the C compiler used.

- Saves the function code setting of the system call into the r10 register.
- Saves the address setting for the return from the system call into the lp register.
- Checks the system call parameters.
- Acquires the address of the system call entry (the value of the hp register + address 0x100).
- Jumps to the system call entry.

If an error is detected as the result of the system call parameter check, the interface library executes the following.

- Saves the error code associated with the detected error into the r10 register.

Figure 11-2 shows an example of the coding of a system call interface library.

Figure 11-2 Example of System Call Interface Library

```
    .text
    .globl  _syscall_name
    .align  2
_syscall_name :
    -- Set the function code to be saved.
    mov     func_code, r10

    -- Parameter check.
    :
    :

    jz      _syscall_err

    -- Acquire the address of the system call entry.
    ld.w    0x100[hp], r12

    -- Jump to the system call entry.
    jmp     [r12]
```

11.6 Extended SVC Handler Interface Library

The main operation of the extended SVC handler interface library is detailed below.
The method of extended SVC handler parameter passing, however, complies with the C compiler being used.

- Saves the function code setting of the extended SVC handler into the r10 register.
- Saves the address setting for the return from the extended SVC handler into the lp register.
- Checks the extended SVC handler parameters.
- Saves the setting of the extended SVC handler parameter area size into the r11 register.

  Exp. In the case of 4 parameters of int type, 0x10 is saved into the r11 register.

- Acquires the address of the extended SVC handler entry (the value of the hp register + address 0x108).
- Jumps to the extended SVC handler entry.

If an error is detected as a result of the extended SVC handler parameter check, the interface library executes the following.

- Saves the error code associated with the detected error into the r10 register.

Figure 11-3 shows an example of coding an extended SVC handler interface library.

Figure 11-3 Example of Extended SVC Handler Interface Library

```
    .text
    .globl  _svchdr_name
    .align  2
_svchdr_name :
    -- Set the function code to be saved.
    mov     func_code, r10

    -- Parameter check.
    :
    :

    jz      _svchdr_err

    -- Set the parameter area size to be saved.
    mov     prm_siz, r11

    -- Acquire the address of the extended SVC handler entry.
    ld.w    0x108[hp], r12

    -- Jump to the extended SVC handler entry.
    jmp     [r12]
```

CHAPTER 12 SYSTEM CALLS

This chapter describes the system calls supported by the RX850 Pro.

12.1 Outline

A system call is a procedure or function for invoking RX850 Pro service routines from the user's processing programs (tasks/non-tasks). The user can use system calls to indirectly manipulate those resources (such as counters and queues) that are managed directly by the RX850 Pro.

The RX850 Pro supports its own system calls as well as the system calls defined in the uITRON 3.0 specifications, thus enhancing the versatility of application systems.

System calls can be classified into the following 7 groups, according to their functions.

- Task management system calls (14)
  These system calls are used to manipulate the status of a task.
  This group provides functions for creating, activating, terminating, and deleting a task, a function for disabling and resuming dispatch processing, a function for changing the task priority, a function for rotating a task ready queue, a function for forcibly releasing a task from the wait state, and a function for referencing the task status.

  cre_tsk, del_tsk, sta_tsk, ext_tsk, exd_tsk, ter_tsk, dis_dsp, ena_dsp, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid

- Task-associated synchronization system calls (7)
  These system calls perform synchronous operations associated with tasks.
  This group provides a function for placing a task in the suspend state and restarting a suspended task, a function for placing a task in the wake-up wait state and waking up a task currently in the wake-up wait state, and another function for canceling a task wake-up request.

  sus_tsk, rsm_tsk, frsm_tsk, slp_tsk, tslp_tsk, wup_tsk, can_wup

- Synchronous communication system calls (25)
  These system calls are used for the synchronization (exclusive control and queuing) and communication between tasks.
  This group provides a function for manipulating semaphores, a function for manipulating events and flags, and a function for manipulating mailboxes.

  cre_sem, del_sem, sig_sem, wai_sem, preq_sem, twai_sem, ref_sem, vget_sid, cre_flg, del_flg, set_flg, clr_flg, wai_flg, pol_flg, twai_flg, ref_flg, vget_fid, cre_mbx, del_mbx, snd_msg, rcv_msg, prcv_msg, trcv_msg, ref_mbx, vget_mid

- Interrupt management system calls (7)
  These system calls perform processing that is dependent on the maskable interrupts.
  This group provides a function for registering an indirectly activated interrupt handler and subsequently canceling the registration, a function for returning from a directly activated interrupt handler, and a function for changing or referencing an interrupt-enabled level.

  def_int, ena_int, dis_int, loc_cpu, unl_cpu, chg_icr, ref_icr

- Memory pool management system calls (8)
  These system calls allocate memory.
  This group provides a function for creating and deleting a memory pool, a function for acquiring and returning a memory block, and a function for referencing the status of a memory pool.

  cre_mpl, del_mpl, get_blk, pget_blk, tget_blk, rel_blk, ref_mpl, vget_pid

- Time management system calls (6)
  These system calls perform processing that is dependent on time.
  This group provides a function for setting or referencing the system clock, a function for placing a task in the timeout wait state, a function for registering a cyclic handler and subsequently canceling the registration, and a function for controlling and referencing the state of a cyclic handler.

  set_tim, get_tim, dly_tsk, def_cyc, act_cyc, ref_cyc

- System management system calls (4)
  These system calls perform processing that varies with the system.
  This group provides a function for acquiring version information, a function for referencing the system status, a function for registering an extended SVC handler and subsequently canceling the registration, and a function for calling an extended SVC handler.

  get_ver, ref_sys, def_svc, viss_svc

12.2 Calling System Calls

System calls issued from processing programs (task/non-task) written in C language are called as C language functions. Their parameters are passed as arguments.

When issuing system calls from processing programs written in assembly language, set parameters and a return address according to the function calling rules of the C compiler, used before calling them with the jarl instruction.

Remark The RX850 Pro declares the prototype of a system call in the stdrx85p.h file. Accordingly, when issuing a system call from a processing program, the following must be coded to include the header file:

```
#include    <stdrx85p.h>
```

12.3 System Call Function Codes

The system calls supported by the RX850 Pro are assigned function codes conforming to the uITRON3.0 specifications. Table 12-1 lists the function codes assigned to system calls.

In the RX850 Pro, a value of 1 or greater is used when registering an extended SVC handler described by the user.

Table 12-1 System Call Function Codes

Function Code	Classification
-256 to -225	RX850 Pro original system calls
-224 to -5	System calls conforming to the uITRON3.0 specifications
-4 to 0	Reserved by the system
1 or more	Extended SVC handler

12.4 Data Types of Parameters

The system calls supported by the RX850 Pro have parameters that are defined based on data types that conform to the uITRON3.0 specifications.

Table 12-2 lists the data types of the parameters specified upon the issuance of a system call.

Table 12-2 Data Types of Parameters

Macro	Data Type	Description
B	signed char	Signed 8-bit integer
H	signed short	Signed 16-bit integer
INT	signed int	Signed 32-bit integer
W	signed long	Signed 32-bit integer
UB	unsigned char	Unsigned 8-bit integer
UH	unsigned short	Unsigned 16-bit integer
UINT	unsigned int	Unsigned 32-bit integer
UW	unsigned long	Unsigned 32-bit integer
VB	signed char	Variable data type value (8 bits)
VH	signed short	Variable data type value (16 bits)
VW	signed long	Variable data type value (32 bits)
*VP	void	Variable data type value (pointer)
(*FP) ( )	void	Processing program start address
BOOL	signed short	Boolean value
FN	signed short	Function code
ID	signed short	Object ID number
BOOL_ID	signed short	Wait task available or not
HNO	signed short	Cyclic handler specification number
ATR	unsigned short	Object attribute
ER	signed long	Error code
PRI	signed short	Task priority
TMO	signed long	Wait time
CYCTIME	signed long	Cyclically activated time interval (residual time)
DLYTIME	signed long	Delay time

12.5 Parameter Value Range

Some of the system call parameters supported by the RX850 Pro have a range of permissible values, while others allow the use of only system reserved specific values.

Table 12-3 lists the ranges of parameter values that can be specified upon the issuance of a system call.

Table 12-3 Ranges of Parameter Values

Parameter Type	Value Range
Object ID number	0x0 to max_cnt[Note1]
Object key ID number	-0x8000 to 0x7FFF[Note2]
Interrupt handler interrupt level	0x0 to 0xF
Specification number of cyclic handler	0x1 to max_cnt
Extended function code of extended SVC handler	0x1 to max_cnt
Object priority	0x1 to max_cnt
Maximum resource count	0x1 to max_cnt
Interrupt enable level of maskable interrupt	0x0 to 0xF
System clock time	0x0 to 0x7FFF FFFF FFFF
Wait time	-0x1 to 0x7FFF FFFF
Delay time	0x0 to 0x7FFF FFFF
Activation time interval of cyclic handler	0x1 to 0x7FFF FFFF
Task stack size	0x0 to 0x7FFF FFFF
Memory pool size	0x1 to 0x7FFF FFFF
Memory block size	0x1 to 0x7FFF FFFF
Message priority	0x1 to 0x7FFF

Note1 max_cnt: The maximum number of objects specified in System maximum value information during system configuration.

Note2 "0x0" cannot be specified for the object key ID number.

12.6 System Call Return Values

The system call return values supported by the RX850 Pro are based on the uITRON3.0 specifications. Table 12-4 lists the system call return values.

Table 12-4 System Call Return Values

Macro	Value	Description
E_OK	0	Normal termination.
E_NOMEM	-10	An area for objects cannot be allocated.
E_NOSPT	-17	A system call with the CF not defined, or an unregistered extended SVC handler was called.
E_RSATR	-24	Invalid object attribute specification.
E_PAR	-33	Invalid parameter specification.
E_ID	-35	Invalid ID number specification.
E_NOEXS	-52	No relevant object exists.
E_OBJ	-63	The status of the specified object is invalid.
E_OACV	-66	An unauthorized ID number was specified.
E_CTX	-69	The state in which the system call is issued is invalid.
E_QOVR	-73	The count exceeded 127.
E_DLT	-81	The target object was deleted.
E_TMOUT	-85	Timeout.
E_RLWAI	-86	A wait state was forcibly canceled by rel_wai.

12.7 System Call Extension

The RX850 Pro supports the extension of system calls (functions coded by users are registered in the nucleus as extended system calls).

No limitations are imposed on those functions registered as extended system calls; standard system calls (system calls supported by the RX850 Pro) can also be included. If, however, standard system calls that can be issued only in the task state are included, the issuance state of the extended system calls is limited to "issuable only from task".

Extended system calls are positioned as user-defined system calls, despite their having properties similar to tasks.

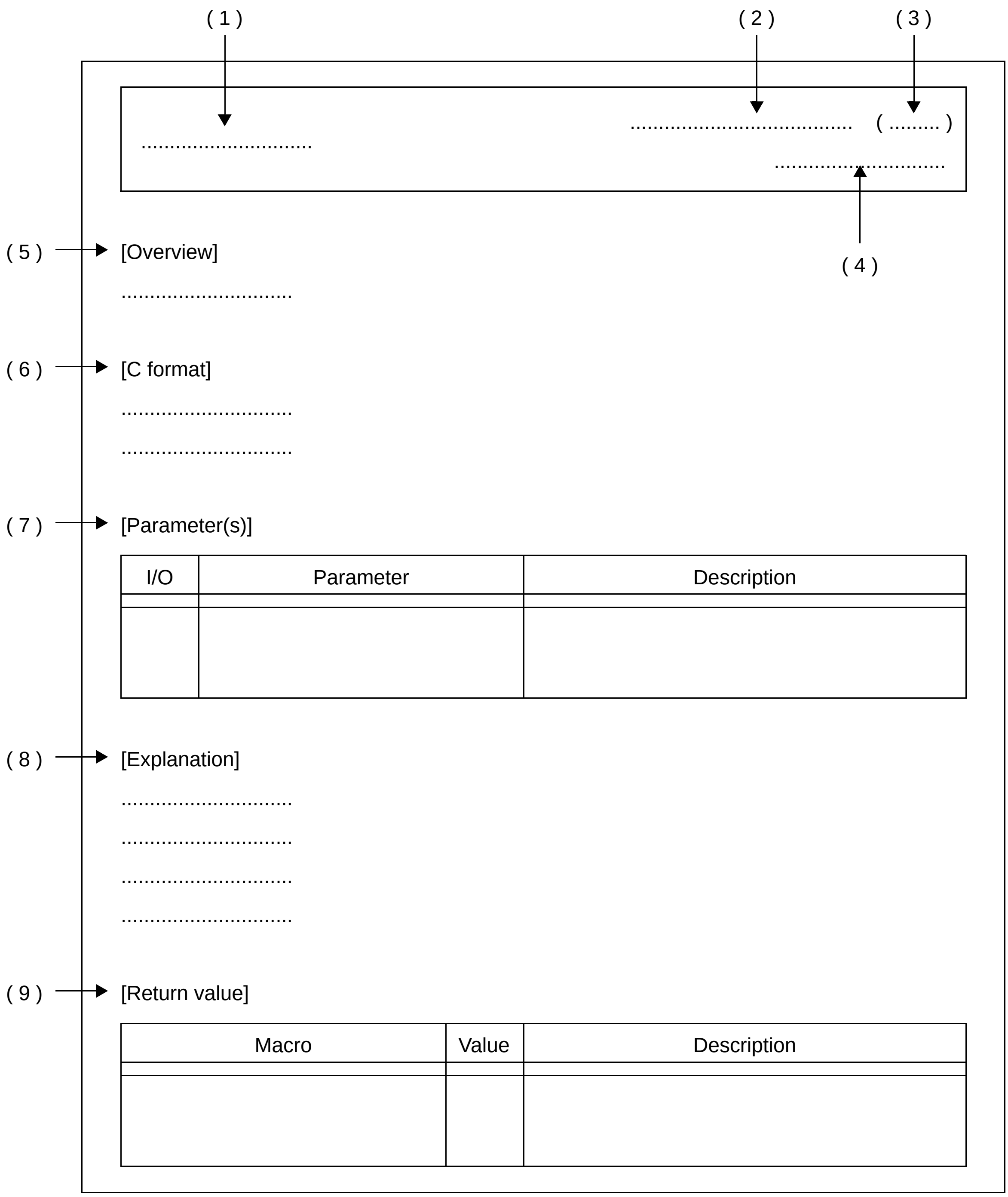
That is, like standard system calls, the scheduler is started upon the termination of processing and an optimum task is selected.

If a standard system call is included in extended system calls, note that control may pass to another task that is currently processing an extended system call because the scheduler is also started upon the termination of a standard system call.

12.8   Explanation of System Calls

The following explains the system calls supported by the RX850 Pro, in the format shown below.

Figure 12-1 System Call Description Format



(1) Name
Indicates the name of the system call.

(2) Semantics
Indicates the source of the name of the system call.

(3) Function code
Indicates the function code of the system call.

(4) Origin of system call
Indicates where the system call can be issued.

Task:	The system call can only be issued from a task.
Non-task:	The system call can only be issued from a non-task (directly activated interrupt handler, indirectly activated interrupt handler and cyclic handler).
Task/Non-task:	The system call can be issued from both a task and a non-task.
Directly activated interrupt handler:	The system call can only be issued from a directly activated interrupt handler.

(5) [Overview]
Outlines the functions of the system call.

(6) [C format]
Indicates the format to be used when describing a system call to be issued in C language.

(7) [Parameter(s)]
System call parameters are explained in the following format.

I/O	Parameter	Description

A: Parameter classification

I... Parameter input to RX850 Pro
O ... Parameter output from RX850 Pro

B: Parameter data type

C: Description of parameter

(8) [Explanation]
Explains the function of a system call.

(9) [Return value]
Indicates a system call's return value using a macro and value.

Return value marked with an asterisk (*):	Value returned by both RX850 Pro having and that not having the parameter check function
Return value not marked with an asterisk (*):	Value returned only by RX850 Pro having the parameter check function

12.8.1 Task management system calls

This section explains the group of system calls that are used to manipulate the task status (task management system calls).

Table 12-5 lists the task management system calls.

Table 12-5 Task Management System Calls

System Call	Function
cre_tsk	Creates another task.
del_tsk	Deletes another task.
sta_tsk	Activates another task.
ext_tsk	Terminates the task which issued the system call.
exd_tsk	Terminates the task which issued the system call, then deletes it.
ter_tsk	Forcibly terminates another task.
dis_dsp	Disables dispatch processing.
ena_dsp	Resumes dispatch processing.
chg_pri	Changes the priority of a task.
rot_rdq	Rotates a task ready queue.
rel_wai	Forcibly releases another task from a wait state.
get_tid	Acquires the ID number of the task that issued the system call.
ref_tsk	Acquires task information.
vget_tid	Acquires the task ID number.

cre_tsk

create task (-17)

Task

[Overview]

Creates a task.

[C format]

- When an ID number is specified

```
#include    <stdrx85p.h>
ER          ercd = cre_tsk ( ID tskid, T_CTSK *pk_ctsk );
```

- When an ID number is not specified

```
#include    <stdrx85p.h>
ER          ercd = cre_tsk ( ID_AUTO, T_CTSK *pk_ctsk, ID *p_tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *tskid*;	Task ID number
I	T_CTSK **pk_ctsk*;	Start address of packet storing task creation information
O	ID **p_tskid*;	Address of area used to store ID number

[Structure of task creation information T_CTSK]

```
typedef struct  t_ctsk {
        VP      exinf;          /*Extended information*/
        ATR     tskatr;         /*Task attribute*/
        FP      task;           /*Task activation address*/
        PRI     itskpri;        /*Task priority at activation (initial priority)*/
        INT     stksz;          /*Task stack size*/
        VP      gp;             /*gp register-specific value for task*/
        VP      tp;             /*tp register-specific value for task*/
        ID      keyid;          /*Task key ID number*/
} T_CTSK;
```

[Explanation]

The RX850 Pro supports 2 types of interfaces for task creation: one in which an ID number is specified for task creation, and another in which an ID number is not specified.

- When an ID number is specified
  A task having the ID number specified by *tskid* is created based on the information specified by *pk_ctsk*.
  The specified task changes from the non-existent state to the dormant state, in which it is managed by the RX850 Pro.

- When an ID number is not specified
  A task is created based on the information specified by *pk_ctsk*.
  The specified task changes from the non-existent state to the dormant state, in which it is managed by the RX850 Pro.
  An ID number is allocated by the RX850 Pro and the allocated ID number is stored in the area specified by *p_tskid*.

The following describes task creation information in detail.

exinf ... Extended information
exinf is an area for storing user-specific information on a specified task. It can be used as necessary by the user.
Information set in exinf can be acquired dynamically by issuing ref_tsk from a processing program (task/non-task).

tskatr ... Task attribute

Bit 0 ... Task language
- TA_ASM (0): Assembly language
- TA_HLNG (1): C language

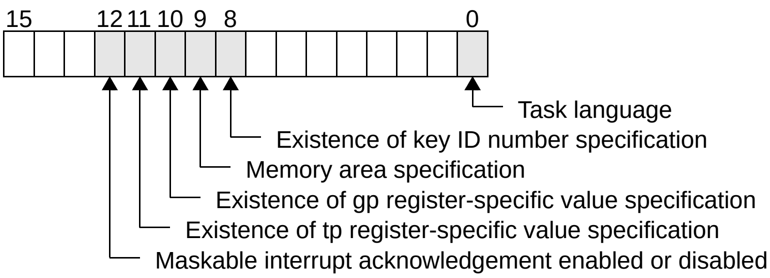
Bit 8 ... Existence of key ID number specification
- TA_KEYID (1): Specifies key ID number

Bit 9 ... Memory area specification
- TA_SPOL0 (0): Secures the stack area from system memory area 0.
- TA_SPOL1 (1): Secures the stack area from system memory area 1.

Bit 10 ... Existence of gp register-specific value specification
- TA_DPID (1): Specifies a gp register-specific value.

Bit 11 ... Existence of tp register-specific value specification
- TA_DPIC (1): Specifies a tp register-specific value.

Bit 12 ... Maskable interrupt acknowledgement enabled or disabled
- TA_ENAINT (0): When a task is activated, the acknowledgement of maskable interrupts is enabled.
- TA_DISINT (1): When a task is activated, the acknowledgement of maskable interrupts is disabled.



task ... Task activation address

itskpri ... Task initial priority (assigned upon activation)

stksz ... Stack size of task (unit: bytes)

gp ... gp register-specific value for task

tp ... tp register-specific value for task

keyid ... Task key ID number

Remark If the value of Bit 8 is not 1 (TA_KEYID), the contents of keyid are meaningless.
If the value of Bit 10 is not 1 (TA_DPID), the contents of gp are meaningless.
If the value of Bit 11 is not 1 (TA_DPIC), the contents of tp are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	An area for task management block cannot be allocated.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute tskatr.

Macro	Value	Description
E_PAR	-33	Invalid parameter specification. - The start address of the packet storing task creation information is invalid (*pk_ctsk* = 0). - Invalid activation address specification (task = 0). - Invalid initial priority specification (*itskpri* ≤ 0, maximum priority < *itskpri*). - Invalid key ID number specification (keyid = 0) (at TA_KEYID attribute specification). - The address of the area used to store the ID number is invalid (*p_tskid* = 0) (When a task is created with no ID number specified).
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_OBJ	-63	A task having the specified ID number has already been created.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

del_tsk

delete task (-18)

Task

[Overview]

Deletes another task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = del_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number

[Explanation]

This system call changes the task specified by *tskid* from the dormant state to the non-existent state.
This releases the target task from the control of the RX850 Pro.
Note that exd_tsk is used when it is necessary for a task to delete itself.

Remark This system call does not queue delete requests. Accordingly, if the target task is not in the dormant state, this system call returns E_OBJ as the return value.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is not in the dormant state.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

sta_tsk

start task (-23)

Task/Non-task

[Overview]

Activates another task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = sta_tsk ( ID tskid, INT stacd );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number
I	`INT stacd;`	Activation code

[Explanation]

This system call changes the task specified by *tskid* from the dormant state to the ready state.

The target task is scheduled by the RX850 Pro.

For stacd, specify the activation code to be passed to the target task. The target task can be manipulated by handling the activation code as if it were a function parameter.

Remark This system call does not queue activation requests. Accordingly, when a target task is not in the dormant state, this system call returns E_OBJ as the return value.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is not in the dormant state.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.

ext_tsk

exit task (-21)

Task

[Overview]

Terminates the task that issued the system call.

[C format]

```
#include    <stdrx85p.h>
void        ext_tsk ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call changes the state of the task from the run state to the dormant state.
The task is excluded from RX850 Pro scheduling.

Remark1 This system call initializes the "task creation information" specified at task creation (at configuration or upon the issuance of cre_tsk).

Remark2 If a task is coded in assembly language, perform coding as follows to terminate the issuing task.

```
jr      _ext_tsk
```

Remark3 If this system call is issued from a non-task or in the dispatch disabled state, its operation is not guaranteed.

Remark4 This system call does not release those resources (memory block, semaphore count, etc.) that were acquired before the termination of the issuing task. Accordingly, the user has to release such resources before issuing this system call.

[Return value]

None.

exd_tsk

exit and delete task (-22)

Task

[Overview]

Terminates the task that issued the system call, then deletes it.

[C format]

```
#include    <stdrx85p.h>
void        exd_tsk ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call changes the task from the run state to the non-existent state.
This releases the task from the control of the RX850 Pro.

Remark1 If this system call is issued from a non-task or in the dispatch disabled state, its operation is not guaranteed.

```
jr      _exd_tsk
```

Remark2 If this system call is issued from a non-task or in the dispatch disabled state, its operation is not guaranteed.

Remark3 This system call does not release those resources (memory block, semaphore count, etc.) that were acquired before the termination of the issuing task. Accordingly, the user has to release such resources before issuing this system call.

[Return value]

None.

ter_tsk	terminate task (-25) Task

[Overview]

Forcibly terminates another task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ter_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID` *`tskid`*;	Task ID number

[Explanation]

This system call forcibly changes the state of the task specified by *tskid* to the dormant state.

Remark1 This system call initializes the "task creation information" specified at task creation (at configuration or upon the issuance of cre_tsk).

Remark2 This system call does not queue termination requests. Accordingly, if a target task is not in the ready, wait, suspend, or wait-suspend state, this system call returns E_NOEXS or E_OBJ as the return value.

Remark3 This system call does not release those resources (memory block, semaphore count, etc.) that were acquired before the termination of the issuing task. Accordingly, the user has to release such resources before issuing this system call.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is the task that issued this system call, or the task is in the dormant state.
E_OACV	-66	An unauthorized ID number (tskid ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

dis_dsp	disable dispatch (-30) Task

[Overview]

Disables dispatch processing.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = dis_dsp ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call disables dispatch processing (task scheduling).
Dispatch processing is disabled until ena_dsp is issued after this system call has been issued.
If a system call such as chg_pri or sig_sem is issued to schedule tasks after this system call is issued but before ena_dsp is issued, the RX850 Pro merely performs operations on a wait queue and delays actual scheduling until ena_dsp is issued, at which time the processing is performed in batch.

Remark1 This system call does not queue disable requests. Accordingly, if this system call has already been issued and dispatch processing has been disabled, no processing is performed and a disable request is not handled as an error.

Remark2 If a system call such as wai_sem and wai_flg is issued, causing the state of the task to change to the wait state after this system call is issued but before ena_dsp is issued, the RX850 Pro returns E_CTX as the return value, regardless of whether the wait conditions are satisfied.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
*E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued after loc_cpu was issued.

ena_dsp

enable dispatch (-29)

Task

[Overview]

Enables dispatch processing.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ena_dsp ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call enables dispatch processing (task scheduling).

If a system call such as chg_pri and sig_sem is issued to schedule tasks after dis_dsp is issued but before this system call is issued, the RX850 Pro merely performs operations on a wait queue and delays actual scheduling until this system call is issued, at which time the processing is performed in batch.

Remark This system call does not queue resume requests. Accordingly, if this system call has already been issued and dispatch processing has been resumed, no processing is performed.
The resume request is not handled as an error.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
*E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued after loc_cpu was issued.

chg_pri

change priority (-27)

Task/Non-task

[Overview]

Changes the priority of a task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = chg_pri ( ID tskid, PRI tskpri );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *tskid*;	Task ID number TSK_SELF (0): Local task Value: Task ID number
I	PRI *tskpri*;	Task priority TPRI_INI (0): Task initial priority Value: Task priority

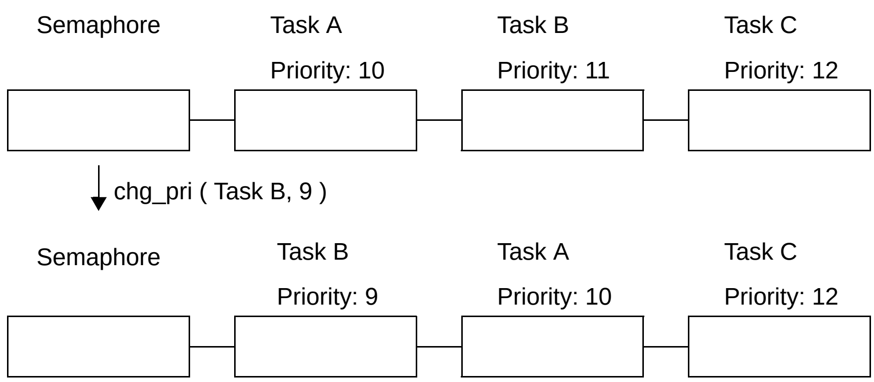
[Explanation]

This system call changes the value of the task priority specified by *tskid* to that specified by *tskpri*.

If the target task is in the run state or the ready state, this system call executes priority change processing and queues the target task at the tail end of the ready queue in accordance with its priority.

Remark1 If the specified task is queued in a wait queue according to its priority, the issue of this system call may change the wait order.

[Example]
When 3 tasks (task A: priority 10, task B: priority 11, task C: priority 12) are placed in a semaphore wait queue according to their priority, and if the priority of task B is changed from 11 to 9, then the wait order of the wait queue changes as shown below.



Remark2 The value specified by *tskpri* is active until the next this system call is issued, or until the target task changes to the dormant state.

Remark3 The task priority in the RX850 Pro becomes higher as its value decreases.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid priority specification (*tskpri* < 0, maximum priority < *tskpri*).
E_ID	-35	Invalid ID number specification. - Maximum number of tasks created < *tskid*. - When this system call was issued from a non-task, TSK_SELF was specified in *tskid*.
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is in the dormant state.
E_OACV	-66	An unauthorized ID number (*tskid* < 0) was specified.

rot_rdq

rotate ready queue (-28)

Task/Non-task

[Overview]

Rotates a task ready queue.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = rot_rdq ( PRI tskpri );
```

[Parameter(s)]

I/O	Parameter	Description
I	PRI *tskpri*;	Task priority TPRI_RUN (0): Priority of task in run state Value: Task priority

[Explanation]

This system call queues the first task in a ready queue to the end of the queue according to the priority specified by *tskpri*.

Remark1 If no task of the specified priority exists in a ready queue, this system call performs no processing. This is not regarded as an error.

Remark2 By issuing this system call at regular intervals, round-robin scheduling can be achieved.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid priority specification (*tskpri* < 0, maximum priority < t*skpri*).

rel_wai

release wait (-31)

Task/Non-task

[Overview]

Forcibly releases another task from the wait state.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = rel_wai ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number

[Explanation]

This system call forcibly releases the task specified by *tskid* from the wait state.

The target task is excluded from a wait queue, and its state changes from the wait state to the ready state, or from the wait-suspend state to the suspend state.

For a task released from the wait state by this system call, E_RLWAI is returned as the return value of the system call (slp_tsk, wai_sem, etc.) that caused transition to the wait state.

Remark This system call does not release the suspend state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is in neither the wait nor wait-suspend state.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.

get_tid	get task identifier (-24) Task/Non-task

[Overview]

Acquires a task ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = get_tid ( ID *p_tskid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`ID *p_tskid;`	Address of area used to store ID number

[Explanation]

This system call stores the ID number of the task that issued this system call in the area specified by *p_tskid*.

Remark If this system call is issued from a non-task, FALSE (0) is stored in the area specified by *p_tskid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The address of the area used to store the ID number is invalid (*p_tskid* = 0).

ref_tsk	refer task status (-20) Task/Non-task

[Overview]

Acquires task information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_tsk ( T_RTSK *pk_rtsk, ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_RTSK *pk_rtsk;	Start address of packet used to store task information
I	ID tskid;	Task ID number TSK_SELF (0): Local task Value: Task ID number

[Structure of task information T_RTSK]

```
typedef struct  t_rtsk {
        VP      exinf;          /*Extended information*/
        PRI     tskpri;         /*Current priority*/
        UINT    tskstat;        /*Task status*/
        UINT    tskwait;        /*Wait cause*/
        ID      wid;            /*ID number of wait object*/
        INT     wupcnt;         /*Number of wake-up requests*/
        INT     suscnt;         /*Number of suspend requests*/
        ID      keyid;          /*Key ID number*/
} T_RTSK;
```

[Explanation]

This system call stores the task information (extended information, current priority, etc.) specified by *tskid* in the packet specified by *pk_rtsk*.

The following describes the task information in detail.

exinf ... Extended information

tskpri ... Current priority

tskstat ... Task state

TTS_RUN (0x1):	Run state
TTS_RDY (0x2):	Ready state
TTS_WAI (0x4):	Wait state
TTS_SUS (0x8):	Suspend state
TTS_WAS (0xc):	Wait-suspend state
TTS_DMT (0x10):	Dormant state

tskwait ... Type of wait state

TTW_SLP (0x1):	Wake-up wait state
TTW_DLY (0x2):	Timeout wait state
TTW_FLG (0x10):	Eventflag wait state
TTW_SEM (0x20):	Resource wait state
TTW_MBX (0x40):	Message wait state
TTW_MPL (0x1000):	Memory block wait state

wid ... ID number of wait object (semaphore, event, flag, etc.)

wupcnt ... Number of wake-up requests

suscnt ... Number of suspend requests

keyid ... Key ID number
FALSE (0): No key ID number specified at creation
Value: Key ID number

Remark1 When the value of tskstat is other than TTS_WAI or TTS_WAS, the contents of tskwait will be undefined.

Remark2 When the value of tskwait is other than TTW_FLG, TTW_SEM, TTW_MBX, or TTW_MPF, the contents of wid will be undefined.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store task information is invalid (*pk_rtsk* = 0).
E_ID	-35	Invalid ID number specification. - Maximum number of tasks created < *tskid*. - When this system call was issued from a non-task, TSK_SELF was specified in *tskid*.
*E_NOEXS	-52	The target task does not exist.
E_OACV	-66	An unauthorized ID number (*tskid* < 0) was specified.

vget_tid

get task Identifier (-248)

Task/Non-task

[Overview]

Acquires a task ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = vget_tid ( ID *p_tskid, ID keyid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`ID *p_tskid;`	Address of area used to store ID number
I	`ID keyid;`	Task key ID number

[Explanation]

This system call stores the task ID number specified by *keyid* in the area specified by *p_tskid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the ID number is invalid (*p_tskid* = 0). - Invalid key ID number specification (*keyid* = 0).
*E_NOEXS	-52	The target task does not exist.

12.8.2 Task-associated synchronization system calls

This section explains the group of system calls that perform the synchronous operations associated with tasks (task-associated synchronization system calls).

Table 12-6 lists the task-associated synchronization system calls.

Table 12-6 Task-Associated Synchronization System Calls

System Call	Function
sus_tsk	Places another task in the suspend state.
rsm_tsk	Restarts a task in the suspend state.
frsm_tsk	Forcibly restarts a task in the suspend state.
slp_tsk	Places the task that issued this system call into the wake-up wait state.
tslp_tsk	Places the task that issued this system call into the wake-up wait state (with timeout).
wup_tsk	Wakes up another task.
can_wup	Invalidates a request to wake up a task.

sus_tsk	suspend task (-33) Task/Non-task

[Overview]

Places another task in the suspend state.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = sus_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number

[Explanation]

This system call issues a suspend request to the task specified by *tskid* (the suspend request counter is incremented by 0x1).

If the target task is in the ready or wait state when this system call is issued, this system call changes the target task from the ready state to the suspend state or from the wait state to the wait-suspend state, and also issues a suspend request (increments the suspend request counter).

Remark The suspend request counter managed by the RX850 Pro consists of 7 bits. Therefore, once the number of suspend requests exceeds 127, this system call returns E_QOVR as the return value without incrementing the suspend request counter.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	Invalid state of the specified task. - The target task is in the dormant state. - The issuing task is specified as the target task when this system call is issued from a task.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.
*E_QOVR	-73	The number of suspend requests exceeded 127.

rsm_tsk

resume task (-35)

Task/Non-task

[Overview]

Restarts a task in the suspend state.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = rsm_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number

[Explanation]

This system call cancels only one of the suspend requests that are issued to the task specified by *tskid* (the suspend request counter is decremented by 0x1).

If the issuance of this system call causes the suspend request counter for the target task to be 0x0, this system call changes the task from the suspend state to the ready state or from the wait-suspend state to the wait state.

Remark This system call does not queue cancel requests. Accordingly, if a target task is not in the suspend or wait-suspend state, this system call returns E_OBJ as the return value without decrementing the suspend request counter.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is not in the suspend or wait-suspend state.
E_OACV	-66	An unauthorized ID number (*tskid* ≤ 0) was specified.

frsm_tsk

force resume task (-36)

Task/Non-task

[Overview]

Forcibly restarts a task in the suspend state.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = frsm_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID tskid;`	Task ID number

[Explanation]

This system call cancels all the suspend requests issued to the task specified by *tskid* (the suspend request counter is set to 0x0).

The target task changes from the suspend state to the read state or from the wait-suspend state to the wait state.

Remark This system call does not queue cancel requests. Accordingly, if a target task is not in the suspend or wait-suspend state, this system call returns E_OBJ as the return value without setting the suspend request counter.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is not in the suspend or wait-suspend state.
E_OACV	-66	An unauthorized ID number (*tskid* $\leq$ 0) was specified.

slp_tsk

sleep task (-38)

Task

[Overview]

Places the task that issued this system call into the wake-up wait state.

[C format]

```
#include     <stdrx85p.h>
ER           ercd = slp_tsk ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call cancels only one of the wake-up requests issued to the task (the wake-up request counter is decremented by 0x1).

If the wake-up request counter for the task is 0x0 when this system call is issued, this system call changes the state of the task from the run state to the wait state (wake-up wait state) without canceling a wake-up request (decrementing the wake-up request counter).

The wake-up wait state is released when wup_tsk or rel_wai is issued. The task changes from the wake-up wait state to the ready state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_RLWAI	-86	The wake-up wait state was forcibly released by rel_wai.

tslp_tsk

sleep task with timeout (-37)

Task

[Overview]

Places the task that issued this system call into the wake-up wait state (with timeout).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = tslp_tsk ( TMO tmout );
```

[Parameter(s)]

I/O	Parameter	Description
I	TMO *tmout*;	Wait time (unit: ms) TMO_POL (0): Quick return TMO_FEVR (-1): Permanent wait Value: Wait time

[Explanation]

This system call cancels only one of the wake-up requests issued to the task (the wake-up request counter is decremented by 0x1).

If the wake-up request counter for the task is 0x0 when this system call is issued, this system call changes the task from the run state to the wait state (wake-up wait state) without canceling a wake-up request (decrementing the wake-up request counter).

Note that the wake-up wait state is canceled if the wait time specified by tmout elapses or if wup_tsk or rel_wai is issued, and the issuing task changes to the ready state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid wait time specification (*tmout* < TMO_FEVR).
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_TMOUT	-85	The wait time has elapsed.
*E_RLWAI	-86	The wake-up wait state was forcibly released by rel_wai.

wup_tsk

wakeup task (-39)

Task/Non-task

[Overview]

Wakes up another task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = wup_tsk ( ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *tskid*;	Task ID number

[Explanation]

This system call issues a wake-up request to the task specified by *tskid* (the wake-up request counter is incremented by 0x1).

If the target task is in the wait state (wake-up wait state) when this system call is issued, this system call changes the task from the wake-up wait state to the ready state without issuing a wake-up request (incrementing the wakeup request counter).

Remark The wake-up request counter managed by the RX850 Pro consists of 7-bits. Therefore, when the number of wake-up requests exceeds 127, this system call returns E_QOVR as the return value without incrementing the wake-up request counter.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of tasks created < *tskid*).
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	Invalid state of the specified task. - The target task is in the dormant state. - The issuing task is specified as the target task when this system call is issued from a task.
E_OACV	-66	An unauthorized ID number (*tskid* $\leq$ 0) was specified.
*E_QOVR	-73	The number of wake-up requests exceeded 127.

can_wup

cancel wakeup task (-40)

Task/Non-task

[Overview]

Invalidates a request to wake up a task.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = can_wup ( INT *p_wupcnt, ID tskid );
```

[Parameter(s)]

I/O	Parameter	Description
O	INT *p_wupcnt;	Address of area used to store the number of wake-up requests
I	ID tskid;	Task ID number TSK_SELF (0): Local task Value: Task ID number

[Explanation]

This system call cancels all the wake-up requests issued to the task specified by *tskid* (the wake-up request counter is set to 0x0).

The number of wake-up requests canceled by this system call is stored in the area specified by *p_wupcnt*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The address of the area used to store the number of wake-up requests is invalid (*p_wupcnt* = 0).
E_ID	-35	Invalid ID number specification. - Maximum number of tasks created < *tskid*. - When this system call was issued from a non-task, TSK_SELF was specified for *tskid*.
*E_NOEXS	-52	The target task does not exist.
*E_OBJ	-63	The target task is in the dormant state.
E_OACV	-66	An unauthorized ID number (*tskid* < 0) was specified.

12.8.3 Synchronous communication system calls

This section explains the group of system calls that are used for synchronization (exclusive control and queuing) and communication between tasks (synchronous communication system calls).

Table 12-7 lists the synchronous communication system calls.

Table 12-7 Synchronous Communication System Calls

System Call	Function
cre_sem	Creates a semaphore.
del_sem	Deletes a semaphore.
sig_sem	Returns resources.
wai_sem	Acquires resources.
preq_sem	Acquires resources (polling).
twai_sem	Acquires resources (with timeout).
ref_sem	Acquires semaphore information.
vget_sid	Acquires a semaphore ID number.
cre_flg	Creates an eventflag.
del_flg	Deletes an eventflag.
set_flg	Sets a bit pattern.
clr_flg	Clears a bit pattern.
wai_flg	Checks a bit pattern.
pol_flg	Checks a bit pattern (polling).
twai_flg	Checks a bit pattern (with timeout).
ref_flg	Acquires eventflag information.
vget_fid	Acquires an eventflag ID number.
cre_mbx	Creates a mailbox.
del_mbx	Deletes a mailbox.
snd_msg	Transmits a message.
rcv_msg	Receives a message.
prcv_msg	Receives a message (polling).
trcv_msg	Receives a message (with timeout).
ref_mbx	Acquires mailbox information.
vget_mid	Acquires a mailbox ID number.

cre_sem

create semaphore (-49)

Task

[Overview]

Creates a semaphore.

[C format]

- When an ID number is specified

```
#include    <stdrx85p.h>
ER          ercd = cre_sem ( ID semid, T_CSEM *pk_csem );
```

- When an ID number is not specified

```
#include    <stdrx85p.h>
ER          ercd = cre_sem ( ID_AUTO, T_CSEM *pk_csem, ID *p_semid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *semid*;	Semaphore ID number
I	T_CSEM **pk_csem*;	Start address of packet containing semaphore creation information
O	ID **p_semid*;	Address of area used to store ID number

[Structure of semaphore creation information T_CSEM]

```
typedef struct  t_csem {
        VP      exinf;          /*Extended information*/
        ATR     sematr;         /*Semaphore attribute*/
        INT     isemcnt;        /*Initial resource count*/
        INT     maxsem;         /*Maximum resource count*/
        ID      keyid;          /*Semaphore key ID number*/
} T_CSEM;
```

[Explanation]

The RX850 Pro provides 2 types of interfaces for semaphore creation: one in which an ID number must be specified, and one in which an ID number is not specified.

- When an ID number is specified
  A semaphore having an ID number specified by *semid* is created based on the information specified by *pk_csem*.

- When an ID number is not specified
  A semaphore is created based on the information specified by *pk_csem*.
  An ID number is allocated by the RX850 Pro and the allocated ID number is stored in the area specified by *p_semid*.

Semaphore creation information is described in detail below.

exinf ... Extended information
An area for storing user-specific information on a target semaphore. The user can use this area as required. Information set in exinf can be dynamically acquired by issuing ref_sem from a processing program (tasks and non-tasks).

sematr ... Semaphore attribute

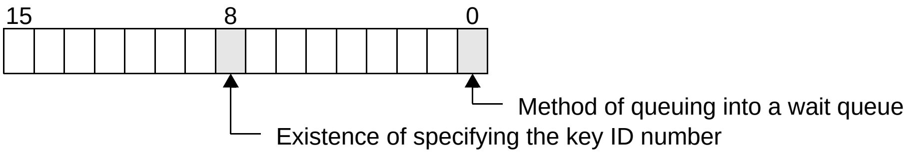
Bit 0 ... Method of queuing into a wait queue

TA_TPRI (0): Priority order

TA_TFIFO (1): FIFO order

Bit 8 ... Existence of specifying the key ID number

TA_KEYID (1): Specifies the key ID number



isemcnt ... Initial resource count

maxsem ... Maximum resource count

keyid ... Semaphore key ID number

Remark If the value of bit 8 is not TA_KEYID in sematr, the contents of keyid are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	The semaphore management block area cannot be secured.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute sematr.
E_PAR	-33	Invalid parameter specification. - The start address of a packet storing semaphore creation information is invalid (*pk_csem* = 0). - The initial resource count is invalid (isemcnt < 0). - The maximum resource count is invalid (maxsem ≤ 0, maxsem < isemcnt). - Invalid key ID number specification (keyid = 0) (when TA_KEYID attribute specified). - The address of the area used to store an ID number is invalid (*p_semid* = 0) (When a semaphore is created without an ID number specified).
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_OBJ	-63	A semaphore having the specified ID number has already been created.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

del_sem	delete semaphore (-50) Task

[Overview]

Deletes a semaphore.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = del_sem ( ID semid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID semid;`	Semaphore ID number

[Explanation]

This system call deletes the semaphore specified by *semid*.
The target semaphore is released from the control of the RX850 Pro.
The task released from the wait state (resource wait state) by this system call has E_DLT returned as the return value of the system call (wai_sem or twai_sem) that initiated transition to the wait state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

sig_sem

signal semaphore (-55)

Task/Non-task

[Overview]

Returns resources.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = sig_sem ( ID semid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID semid;`	Semaphore ID number

[Explanation]

This system call returns resources to the semaphore specified by *semid* (the semaphore counter is incremented by 0x1).

If tasks are queued in the wait queue of the target semaphore when this system call is issued, this system call passes the resources to the relevant task (the first task in the wait queue) without returning the resources (incrementing the semaphore counter).

Consequently, the relevant task is removed from the wait queue, and its state changes from the wait state (resource wait state) to the ready state, or from the wait-suspend state to the suspend state.

Remark The semaphore counter managed by the RX850 Pro counts up to the maximum resource count that can be acquired as specified at the time it is created. Therefore, when the number of resources exceeds the maximum resource count, by issuing this system call, E_QOVR is returned as the return value without incrementing the semaphore counter.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
*E_QOVR	-73	The resource count exceeded the maximum resource count specified at creation.

wai_sem

wait on semaphore (-53)

Task

[Overview]

Acquires resources.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = wai_sem ( ID semid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID semid;`	Semaphore ID number

[Explanation]

This system call acquires resources from the semaphore specified by *semid* (the semaphore counter is decremented by 0x1).

When this system call is issued, if no resource can be acquired from a target semaphore (when there are no free resources), this system call places the task in the wait queue of the specified semaphore, then changes it from the run state to the wait state (resource wait state).

The resource wait state is released upon the issuance of sig_sem, del_sem, or rel_wai, and the task returns to the ready state.

Remark When a task queues in the wait queue of the target semaphore, it is executed in the order (FIFO order or priority order) specified when that semaphore was created (at configuration or when cre_seml was issued).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_DLT	-81	The specified semaphore was deleted by del_sem.
*E_RLWAI	-86	The resource wait state was forcibly released by rel_wai.

preq_sem	poll and request semaphore (-107) Task/Non-task

[Overview]

Acquires resources (polling).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = preq_sem ( ID semid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID semid;`	Semaphore ID number

[Explanation]

This system call acquires resources from the semaphore specified by *semid* (the semaphore counter is decremented by 0x1).

When this system call is issued, if no resource can be acquired from a target semaphore (when there are no free resources), this system call returns E_TMOUT as the return value.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
*E_TMOUT	-85	The resource count for the target semaphore is 0x0.

twai_sem

wait on semaphore with timeout (-171)

Task

[Overview]

Acquires resources (with timeout).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = twai_sem ( ID semid, TMO tmout );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *semid*;	Semaphore ID number
I	TMO *tmout*;	Wait time (unit: ms) TMO_POL (0): Quick return TMO_FEVR (-1): Permanent wait Value: Wait time

[Explanation]

This system call acquires resources from the semaphore specified by *semid* (the semaphore counter is decremented by 0x1).

When this system call is issued, if no resource can be acquired from a target semaphore (when there are no free resources), this system call places the task in the wait queue of the target semaphore, then changes it from the run state to the wait state (resource wait state).

The resource wait state is released when the wait time specified by *tmout* elapses or when sig_sem, del_sem, or rel_wai is issued, at which time it changes to the ready state.

Remark The task is queued into the wait queue of a target semaphore in the order (FIFO order or priority order) specified when the semaphore was created (at configuration or upon the issuance of cre_sem).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid wait time specification (*tmout* < TMO_FEVR).
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* ≤ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_DLT	-81	A target semaphore was deleted by del_sem.
*E_TMOUT	-85	Wait time elapsed.

Macro	Value	Description
*E_RLWAI	-86	The resource wait state was forcibly released by rel_wai.

ref_sem

refer semaphore status (-52)

Task/Non-task

[Overview]

Acquires semaphore information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_sem ( T_RSEM *pk_rsem, ID semid );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_RSEM *pk_rsem;	Start address of packet used to store semaphore information
I	ID semid;	Semaphore ID number

[Structure of semaphore information T_RSEM]

```
typedef struct  t_rsem {
        VP      exinf;          /*Extended information*/
        BOOL_ID wtsk;           /*Existence of waiting task*/
        INT     semcnt;         /*Current resource count*/
        INT     maxsem;         /*Maximum resource count*/
        ID      keyid;          /*Key ID number*/
} T_RSEM;
```

[Explanation]

This system call stores the semaphore information (extended information, existence of waiting task, etc.) for the semaphore specified by *semid* in the packet specified by *pk_rsem*.

Semaphore information is described in detail below.

exinf ... Extended information

wtsk ... Existence of waiting task
FALSE (0): There is no waiting task
Value: ID number of first task in wait queue

semcnt ... Current resource count

maxsem ... Maximum resource count specified at creation

keyid ... Key ID number
FALSE (0): No key ID number specified at creation
Value: Key ID number

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store semaphore information is invalid (*pk_rsem* = 0).

Macro	Value	Description
E_ID	-35	Invalid ID number specification (maximum number of semaphores created < *semid*).
*E_NOEXS	-52	The target semaphore does not exist.
E_OACV	-66	An unauthorized ID number (*semid* $\leq$ 0) was specified.

vget_sid

get semaphore identifier (-246)

Task/Non-task

[Overview]

Acquires the semaphore ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = vget_sid ( ID *p_semid, ID keyid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`ID *p_semid;`	Address of area used to store ID number
I	`ID keyid;`	Semaphore key ID number

[Explanation]

This system call stores the semaphore ID number specified by *keyid* in the area specified by *p_semid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the ID number is invalid (*p_semid* = 0). - Invalid key ID number specification (*keyid* = 0).
*E_NOEXS	-52	The target semaphore does not exist.

cre_flg	create eventflag (-41) Task

[Overview]

Creates an eventflag.

[C format]

- When an ID number is specified

```
#include    <stdrx85p.h>
ER          ercd = cre_flg ( ID flgid, T_CFLG *pk_cflg );
```

- When an ID number is not specified

```
#include    <stdrx85p.h>
ER          ercd = cre_flg ( ID_AUTO, T_CFLG *pk_cflg, ID *p_flgid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *flgid*;	Eventflag ID number
I	T_CFLG **pk_cflg*;	Start address of packet storing eventflag creation information
O	ID **p_flgid*;	Address of area used to store ID number

[Structure of eventflag creation information T_CFLG]

```
typedef struct  t_cflg {
        VP      exinf;          /*Extended information*/
        ATR     flgatr;         /*Eventflag attribute*/
        UINT    iflgptn;        /*Initial bit pattern of eventflag*/
        ID      keyid;          /*Eventflag key ID number*/
} T_CFLG;
```

[Explanation]

The RX850 Pro provides 2 types of interfaces for eventflag creation: one in which an ID number must be specified and one in which an ID number is not specified.

- When an ID number is specified
  An eventflag having the ID number specified by *flgid* is created based on the information specified by *pk_cflg*.

- When an ID number is not specified
  An eventflag is created based on the information specified by *pk_cflg*.
  An ID number is allocated by the RX850 Pro and the allocated ID number is stored in the area specified by *p_flgid*.

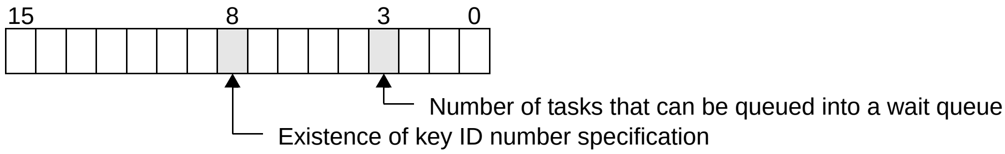
Eventflag creation information is described in detail below.

exinf ... Extended information
exinf is an area used for storing user-specific information on a target eventflag. The user can use this area as required.
Information set in exinf can be dynamically acquired by issuing ref_flg from a processing program (task or non-task).

flgatr ... Eventflag attribute

Bit 3 ... Number of tasks that can be queued into a wait queue
TA_WSGL (0): One task only
TA_WMUL (1): 2 or more tasks

Bit 8 ... Existence of key ID number specification
TA_KEYID (1): Key ID number specified



iflgptn ... Initial bit pattern of eventflag

keyid ... Eventflag key ID number

Remark If the value of bit 8 is not TA_KEYID in flgatr, the contents of keyid are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	The eventflag management block area cannot be secured.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute flgatr.
E_PAR	-33	Invalid parameter specification. - The start address of the packet storing eventflag creation information is invalid (*pk_cflg* = 0). - Invalid key ID number specification (keyid = 0) (when TA_KEYID attribute specified). - The address of the area used to store the ID number is invalid (*p_flgid* = 0) (When an eventflag is created with no ID number specified).
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_OBJ	-63	An eventflag having the specified ID number has already been created.
E_OACV	-66	An unauthorized ID number (*flgid* $\leq$ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

del_flg

delete eventflag (-42)

Task

[Overview]

Deletes an eventflag.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = del_flg ( ID flgid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID flgid;`	Eventflag ID number

[Explanation]

This system call deletes the eventflag specified by *flgid*.

The target eventflag is released from the control of the RX850 Pro.

The task released from the wait state (eventflag wait state) by this system call has E_DLT returned as the return value of the system call (wai_flg or twai_flg) that initiated transition to the wait state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
E_OACV	-66	An unauthorized ID number (*flgid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

set_flg

set eventflag (-48)

Task/Non-task

[Overview]

Sets a bit pattern.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = set_flg ( ID flgid, UINT setptn );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *flgid*;	Eventflag ID number
I	UINT *setptn*;	Bit pattern to be set (32-bit width)

[Explanation]

This system call executes a logical OR between the bit pattern specified by *flgid* and that specified by *setptn*, and sets the result in the specified eventflag.

For example, when this system call is issued, if the target eventflag's bit pattern is B'1100 and the bit pattern specified by *setptn* is B'1010, the bit pattern of the target eventflag becomes B'1110.

When this system call is issued, if the wait condition for a task queued in the wait queue of the target eventflag is satisfied, the task is removed from the wait queue.

Consequently, the relevant task changes from the wait state (eventflag wait state) to the ready state, or from the wait-suspend state to the suspend state.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
E_OACV	-66	An unauthorized ID number (*flgid* ≤ 0) was specified.

clr_flg

clear eventflag (-47)

Task/Non-task

[Overview]

Clears a bit pattern.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = clr_flg ( ID flgid, UINT clrptn );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID flgid;`	Eventflag ID number
I	`UINT clrptn;`	Bit pattern to be cleared (32-bit width)

[Explanation]

This system call executes a logical AND between the bit pattern specified by *flgid* and that specified by *clrptn*, and sets the result in the specified eventflag.

For example, when this system call is issued, if the target eventflag's bit pattern is B'1100 and the bit pattern specified by *clrptn* is B'1010, the target eventflag's bit pattern becomes B'1000.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
E_OACV	-66	An unauthorized ID number (*flgid* ≤ 0) was specified.

wai_flg

wait eventflag (-46)

Task

[Overview]

Checks a bit pattern.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = wai_flg ( UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode );
```

[Parameter(s)]

I/O	Parameter	Description
O	`UINT *p_flgptn;`	Address of area used to store bit pattern when condition is satisfied
I	`ID flgid;`	Eventflag ID number
I	`UINT waiptn;`	Request bit pattern (32-bit width)
I	`UINT wfmode;`	Wait condition or condition satisfaction TWF_ANDW (0): AND wait TWF_ORW (2): OR wait TWF_CLR (1): Bit pattern is cleared

[Explanation]

This system call checks whether a bit pattern that satisfies the request bit pattern specified by *waiptn*, as well as the wait condition specified by *wfmode*, is set in the eventflag specified by *flgid*.

If a bit pattern satisfying the wait condition is set in the target eventflag, this system call stores the bit pattern of the eventflag in the area specified by *p_flgptn*.

When this system call is issued, if the bit pattern of the target eventflag does not satisfy the wait condition, this system call queues the task at the end of the wait queue for the target eventflag, then changes it from the run state to the wait state (eventflag wait state).

The eventflag wait state is released when a bit pattern satisfying the wait condition is set by set_flg, or when del_flg or rel_wai is issued, at which time it changes to the ready state.

The specification format for *wfmode* is shown below.

- *wfmode* = TWF_ANDW
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.

- *wfmode* = (TWF_ANDW | TWF_CLR)
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.
  If the wait condition is satisfied, the bit pattern for the target eventflag is cleared (B'0000 is set).

- *wfmode* = TWF_ORW
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.

- *wfmode* = (TWF_ORW | TWF_CLR)
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.
  If the wait condition is satisfied, the bit pattern of the target eventflag is cleared (B'0000 is set).

Remark1 The RX850 Pro specifies the number of tasks that can be queued into the wait queue of an eventflag at creation (at configuration or upon the issuance of cre_flg).

TA_WSGL attribute: Only one task can be queued.
TA_WMUL attribute: 2 or more tasks can be queued.

For this reason, if this system call is issued for the eventflag having the TA_WSGL attribute for which waiting tasks are already queued, this system call returns E_OBJ as the return value without performing bit pattern checking.

Remark2 If the eventflag wait state is forcibly released by issuing del_flg or rel_wai, the contents of the area specified by *p_flgptn* will be undefined.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store a bit pattern when a condition is satisfied is invalid (*p_flgptn* = 0). - Invalid specification of request bit pattern (*waiptn* = 0). - Invalid specification of wait condition or condition satisfaction parameter *wfmode*.
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
*E_OBJ	-63	This system call was issued for the eventflag having the TA_WSGL attribute for which waiting tasks were already queued.
E_OACV	-66	An unauthorized ID number ($flgid \leq 0$) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued from the dispatch disabled state.
*E_DLT	-81	The target eventflag was deleted by del_flg.
*E_RLWAI	-86	The eventflag wait state was forcibly released by rel_wai.

pol_flg

poll eventflag (-106)

Task/Non-task

[Overview]

Checks a bit pattern (polling).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = pol_flg ( UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode );
```

[Parameter(s)]

I/O	Parameter	Description
O	UINT *p_flgptn;	Address of area used to store bit pattern when condition is satisfied
I	ID flgid;	Eventflag ID number
I	UINT waiptn;	Request bit pattern (32-bit width)
I	UINT wfmode;	Wait condition or condition satisfaction TWF_ANDW (0): AND wait TWF_ORW (2): OR wait TWF_CLR (1): Bit pattern is cleared

[Explanation]

This system call checks whether a bit pattern satisfying both the request bit pattern specified by *waiptn* and the wait condition specified by *wfmode* is set in the eventflag specified by *flgid*.

If a bit pattern satisfying the wait condition is set in the target eventflag, this system call stores the bit pattern of the eventflag into the area specified by *p_flgptn*.

When this system call is issued, if the bit pattern of the target eventflag does not satisfy the wait condition, this system call returns E_TMOUT as the return value.

The *wfmode* specification format is shown below.

- *wfmode* = TWF_ANDW
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.

- *wfmode* = (TWF_ANDW | TWF_CLR)
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.
  If the wait condition is satisfied, the bit pattern for the target eventflag is cleared (B'0000 is set).

- *wfmode* = TWF_ORW
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.

- *wfmode* = (TWF_ORW | TWF_CLR)
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.
  If the wait condition is satisfied, the bit pattern of the target eventflag is cleared (B'0000 is set).

Remark The RX850 Pro specifies the number of tasks that can be queued into the wait queue of an eventflag at creation (at configuration or upon the issuance of cre_flg).

TA_WSGL attribute: Only one task can be queued.
TA_WMUL attribute: 2 or more tasks can be queued.

For this reason, if this system call is issued for an eventflag having the TA_WSGL attribute for which waiting tasks are already queued, this system call returns E_OBJ as the return value without performing bit pattern checking.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store a bit pattern when a condition is satisfied is invalid (*p_flgptn* = 0). - Invalid specification of request bit pattern (*waiptn* = 0). - Invalid specification of wait condition or condition satisfaction parameter *wfmode*.
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
*E_OBJ	-63	This system call was issued for the eventflag of TA_WSGL attribute for which waiting tasks are already queued.
E_OACV	-66	An unauthorized ID number (*flgid* ≤ 0) was specified.
*E_TMOUT	-85	The bit pattern of the target eventflag does not satisfy the wait condition.

twai_flg

wait eventflag with timeout (-170)

Task

[Overview]

Checks a bit pattern (with timeout).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = twai_flg ( UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO
tmout );
```

[Parameter(s)]

I/O	Parameter	Description
O	UINT *p_flgptn;	Address of area used to store bit pattern when condition is satisfied
I	ID flgid;	Eventflag ID number
I	UINT waiptn;	Request bit pattern (32-bit width)
I	UINT wfmode;	Wait condition or condition satisfaction TWF_ANDW (0): AND wait TWF_ORW (2): OR wait TWF_CLR (1): Bit pattern is cleared
I	TMO tmout;	Wait time (unit: ms) TMO_POL (0): Quick return TMO_FEVR (-1): Permanent wait Value: Wait time

[Explanation]

This system call checks whether a bit pattern satisfying both the request bit pattern specified by *waiptn* and the wait condition specified by *wfmode* is set in the eventflag specified by flgid.

If a bit pattern satisfying the wait condition is set in the target eventflag, this system call stores the bit pattern of the eventflag in the area specified by *p_flgptn*.

Upon the issuance of this system call, if the bit pattern of the target eventflag does not satisfy the wait condition, this system call queues the task at the end of the wait queue for the target eventflag, then changes it from the run state to the wait state (eventflag wait state).

The eventflag wait state is released upon the elapse of the wait time specified by *tmout*, when a bit pattern satisfying the wait condition is set by set_flg, or when del_flg or rel_wai is issued, at which time the task returns to the ready state.

The *wfmode* specification format is shown below.

- *wfmode* = TWF_ANDW
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.

- *wfmode* = (TWF_ANDW | TWF_CLR)
  This system call checks whether all the bits of *waiptn* that are set to 1 are set in the target eventflag.
  If the wait condition is satisfied, the bit pattern for the target eventflag is cleared (B'0000 is set).

- *wfmode* = TWF_ORW
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.

- *wfmode* = (TWF_ORW | TWF_CLR)
  This system call checks whether at least one of the bits of *waiptn* that are set to 1 is set in the target eventflag.
  If the wait condition is satisfied, the bit pattern of the target eventflag is cleared (B'0000 is set).

Remark1 The RX850 Pro specifies the number of tasks that can be queued into the wait queue of the eventflag at creation (at configuration or upon the issuance of cre_flg).

TA_WSGL attribute: Only one task can be queued.
TA_WMUL attribute: 2 or more tasks can be queued.

For this reason, if this system call is issued for an eventflag having the TA_WSGL attribute for which waiting tasks are already queued, this system call returns E_OBJ as the return value without performing bit pattern checking.

Remark2 If the eventflag wait state is forcibly released by del_flg or rel_wai, the contents of the area specified by *p_flgptn* will be undefined.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the bit pattern when the condition is satisfied is invalid (*p_flgptn* = 0). - The specification of the request bit pattern is invalid (*waiptn* = 0). - The specification of the wait condition or condition satisfaction parameter *wfmode* is invalid. - Invalid wait time specification (*tmout* < TMO_FEVR).
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).
*E_NOEXS	-52	The target eventflag does not exist.
*E_OBJ	-63	This system call was issued for the eventflag having the TA_WSGL attribute in which waiting tasks were already queued.
E_OACV	-66	An unauthorized ID number (*flgid* ≤ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued from the dispatch disabled state.
*E_DLT	-81	The specified eventflag was deleted by del_flg.
*E_TMOUT	-85	Wait time elapsed.
*E_RLWAI	-86	The eventflag wait state was forcibly released by rel_wai.

ref_flg

refer eventflag status (-44)

Task/Non-task

[Overview]

Acquires eventflag information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_flg ( T_RFLG *pk_rflg, ID flgid );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_RFLG *pk_rflg;	Start address of packet used to store eventflag information
I	ID flgid;	Eventflag ID number

[Structure of eventflag information T_RFLG]

```
typedef struct  t_rflg {
        VP      exinf;          /*Extended information*/
        BOOL_ID wtsk;           /*Existence of waiting task*/
        UINT    flgptn;         /*Current bit pattern*/
        ID      keyid;          /*Key ID number*/
} T_RFLG;
```

[Explanation]

This system call stores the eventflag information (extended information, existence of waiting task, etc.) for the eventflag specified by *flgid* in the packet specified by *pk_rflg*.

Eventflag information is described in detail below.

exinf ... Extended information

wtsk ... Existence of waiting task
FALSE (0): There is no waiting task.
Value: ID number of first task in wait queue

flgptn ... Current bit pattern

keyid ... Key ID number
FALSE (0): No key ID number specified at generation
Value: Key ID number[Return value]

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store eventflag information is invalid (*pk_rflg* = 0).
E_ID	-35	Invalid ID number specification (maximum number of eventflags created < *flgid*).

Macro	Value	Description
*E_NOEXS	-52	The target eventflag does not exist.
E_OACV	-66	An unauthorized ID number (*flgid* $\leq$ 0) was specified.

vget_fid

get eventflag identifier (-247)

Task/Non-task

[Overview]

Acquires the eventflag ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = vget_fid ( ID *p_flgid, ID keyid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`ID *p_flgid;`	Address of area used to store ID number
I	`ID keyid;`	Eventflag key ID number

[Explanation]

This system call stores the eventflag ID number specified by *keyid* in the area specified by *p_flgid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the ID number is invalid (*p_flgid* = 0). - Invalid key ID number specification (*keyid* = 0).
*E_NOEXS	-52	The target eventflag does not exist.

cre_mbx	create mailbox (-57) Task

[Overview]

Creates a mailbox.

[C format]

- When an ID number is specified

```
#include    <stdrx85p.h>
ER          ercd = cre_mbx ( ID mbxid, T_CMBX *pk_cmbx );
```

- When an ID number is not specified

```
#include    <stdrx85p.h>
ER          ercd = cre_mbx ( ID_AUTO, T_CMBX *pk_cmbx, ID *p_mbxid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *mbxid*;	Mailbox ID number
I	T_CMBX **pk_cmbx*;	Start address of packet used to store mailbox creation information
O	ID **p_mbxid*;	Address of area used to store ID number

[Structure of mailbox creation information T_CMBX]

```
typedef struct  t_cmbx {
        VP      exinf;          /*Extended information*/
        ATR     mbxatr;         /*Mailbox attribute*/
        ID      keyid;          /*Mailbox key ID number*/
} T_CMBX;
```

[Explanation]

The RX850 Pro provides 2 types of interfaces for mailbox creation: one in which an ID number must be specified for mailbox creation, and one in which an ID number is not specified.

- When an ID number is specified
  A mailbox having the ID number specified by *mbxid* is created based on the information specified by *pk_cmbx*.

- When an ID number is not specified
  A mailbox is created based on the information specified by *pk_cmbx*.
  An ID number is allocated by the RX850 Pro. The allocated ID number is stored in the area specified by *p_mbxid*.

Mailbox creation information is described in detail below.

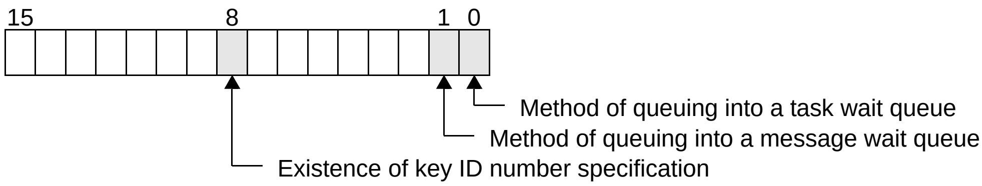
exinf ... Extended information
exinf is an area used for storing user-specific information on the target mailbox. The user can use this area as required.
Information set in exinf can be dynamically acquired by issuing ref_mbx from a processing program (task/non-task).

mbxatr ... Mailbox attribute

Bit 0 ... Method of queuing into a task wait queue
TA_TPRI (0): Priority order
TA_TFIFO (1): FIFO order

Bit 1 ... Method of queuing into a message wait queue
TA_MPRI (0): Priority order
TA_MFIFO (1): FIFO order

Bit 8 ... Existence of key ID number specification
TA_KEYID (1): Key ID number specified



keyid ... Mailbox key ID number

Remark If the value of bit 8 is not TA_KEYID in mbxatr, the contents of keyid are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	The mailbox management block area cannot be secured.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute mbxatr.
E_PAR	-33	Invalid parameter specification. - The start address of the packet storing the mailbox creation information is invalid (*pk_cmbx* = 0). - The specification of the key ID number is invalid (keyid = 0) (when TA_KEYID specified). - The address of the area used to store the ID number is invalid (*p_mbxid* = 0) (When a mailbox is created without an ID number specified)..
E_ID	-35	IInvalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_OBJ	-63	A mailbox having the specified ID number has already been created.
E_OACV	-66	An unauthorized ID number (mbxid ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

del_mbx

delete mailbox (-58)

Task

[Overview]

Deletes a mailbox

[C format]

```
#include        <stdrx85p.h>
ER              ercd = del_mbx ( ID mbxid );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *mbxid*;	Mailbox ID number

[Explanation]

This system call deletes the mailbox specified by *mbxid*.

The target mailbox is released from the control of the RX850 Pro.

The task released from the wait state (message wait state) by this system call has E_DLT returned as the return value of the system call (rcv_msg or trcv_msg) that instigated the transition to the wait state.

Remark When this system call is issued, any message using a memory block acquired from a memory pool is queued into the message wait queue of the target mailbox, and the message (memory block) is then returned to the memory pool.
For this reason, if this system call uses an area other than memory blocks acquired from the memory pool, operation is not guaranteed. This system call should therefore not be issued in the above case.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	The target mailbox does not exist.
E_OACV	-66	An unauthorized ID number (*mbxid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

snd_msg	send message (-63) Task/Non-task

[Overview]

Transmits a message.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = snd_msg ( ID mbxid, T_MSG *pk_msg );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *mbxid*;	Mailbox ID number
I	T_MSG **pk_msg*;	Start address of packet used to store a message

[Structure of message T_MSG]

```
typedef struct  t_msg {
        VW      msgrfu;         /*Message management area*/
        PRI     msgpri;         /*Message priority*/
        VB      msgcont[];      /*Message body*/
} T_MSG;
```

[Explanation]

This system call transmits the message specified in *pk_msg* to the mailbox specified in *mbxid* (queues the message into a message wait queue).

When this system call is issued, if a task is queued into the task wait queue of the target mailbox, this system call passes the message to the task (first task in the task wait queue) without performing message queuing.

Consequently, the relevant task is removed from the task wait queue, and its state changes from the wait state (message wait state) to the ready state, or from the wait-suspend state to the suspend state.

Remark1 When a message queues in the message wait queue of the target mailbox, it is executed in the order (FIFO order or priority order) specified when that mailbox was generated (at configuration or when cre_mbx was issued).

Remark2 The RX850 Pro uses the first 4 bytes (message management area msgrfu) of a message as a link area for enabling queuing into a message wait queue. Accordingly, transmitting a message to the target mailbox requires that 0x0 be set in msgrfu before issuing snd_msg.
If a value other than 0x0 is set in msgrfu when snd_msg is issued, the RX850 Pro recognizes that the relevant message is already queued into a message wait queue, and this system call returns E_OBJ as the return value without transmitting the message.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store the message is invalid (*pk_msg* = 0).

Macro	Value	Description
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	The target mailbox does not exist.
E_OBJ	-63	The area specified for a message is already being used for messages.
E_OACV	-66	An unauthorized ID number (*mbxid* ≤ 0) was specified.

rcv_msg

receive message from mailbox (-61)

Task

[Overview]

Receives a message.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = rcv_msg ( T_MSG **ppk_msg, ID mbxid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`T_MSG **ppk_msg;`	Address of area used to store start address of message
I	`ID mbxid;`	Mailbox ID number

[Explanation]

This system call receives a message from the mailbox specified by *mbxid* and stores its start address in the area specified by *ppk_msg*.

When this system call is issued, if a message cannot be received from the target mailbox (when no message exists in a message wait queue), this system call queues the task into the task wait queue of the target mailbox, then changes its state from the run state to the wait state (message wait state).

The message wait state is released when snd_msg, del_mbx, or rel_wai is issued, and the task returns to the ready state.

Remark When a task queues in the task wait queue of the target mailbox, it is executed in the order (FIFO order or priority order) specified when that mailbox was created (at configuration or when cre_mbx was issued).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The address of the area used to store the start address of a message is invalid (*ppk_msg* = 0).
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	The target mailbox does not exist.
E_OACV	-66	An unauthorized ID number (*mbxid* $\leq$ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued from the dispatch disabled state.
*E_DLT	-81	The target mailbox was deleted by del_mbx.
*E_RLWAI	-86	The message wait state was forcibly released by rel_wai.

prcv_msg

poll and receive message from mailbox (-108)

Task/Non-task

[Overview]

Receives a message (polling).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = prcv_msg ( T_MSG **ppk_msg, ID mbxid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`T_MSG **ppk_msg;`	Address of area used to store the start address of a message
I	`ID mbxid;`	Mailbox ID number

[Explanation]

This system call receives a message from the mailbox specified by *mbxid* and stores its start address in the area specified by *ppk_msg*.

When this system call is issued, if a message cannot be received from the target mailbox (when no message exists in the message wait queue), E_TMOUT is returned as the return value.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The address of the area used to store the start address of the message is invalid (*ppk_msg* = 0).
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	A target mailbox does not exist.
E_OACV	-66	An unauthorized ID number (*mbxid* ≤ 0) was specified.
*E_TMOUT	-85	No message exists in the target mailbox.

trcv_msg	receive message from mailbox with timeout (-172) Task

[Overview]

Receives a message (with timeout).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = trcv_msg ( T_MSG **ppk_msg, ID mbxid, TMO tmout );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_MSG ***ppk_msg;*	Address of area used to store start address of message
I	ID *mbxid;*	Mailbox ID number
I	TMO *tmout;*	Wait time (unit: ms) TMO_POL (0): Quick return TMO_FEVR (-1): Permanent wait Value: Wait time

[Explanation]

This system call receives a message from the mailbox specified by *mbxid* and stores its start address in the area specified by *ppk_msg*.

When this system call is issued, if a message cannot be received from the target mailbox (when no message exists in the message wait queue), this system call queues the task into the task wait queue of the target mailbox, then changes its state from the run state to the wait state (message wait state).

The message wait state is released when the wait time specified by *tmout* elapses or when snd_msg, del_mbx, or rel_wai is issued, and the task returns to the ready state.

Remark When a task queues in the task wait queue of the target mailbox, it is executed in the order (FIFO order or priority order) specified when that mailbox was created (at configuration or when cre_mbx was issued).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The address of the area used to store the start address of a message is invalid (*ppk_msg* = 0).
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	The target mailbox does not exist.
E_OACV	-66	An unauthorized ID number (*mbxid* $\leq$ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued from the dispatch disabled state.

Macro	Value	Description
*E_DLT	-81	The specified mailbox was deleted by del_mbx.
*E_TMOUT	-85	The wait time has elapsed.
*E_RLWAI	-86	The message wait state was forcibly released by rel_wai.

ref_mbx	refer mailbox status (-60) Task/Non-task

[Overview]

Acquires mailbox information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_mbx ( T_RMBX *pk_rmbx, ID mbxid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`T_RMBX *pk_rmbx;`	Start address of packet used to store mailbox information
I	`ID mbxid;`	Mailbox ID number

[Structure of mailbox information T_RMBX]

```
typedef struct  t_rmbx {
        VP      exinf;          /*Extended information*/
        BOOL_ID wtsk;           /*Existence of waiting task*/
        T_MSG   *pk_msg;        /*Existence of waiting message*/
        ID      keyid;          /*Key ID number*/
} T_RMBX;
```

[Explanation]

This system call stores mailbox information (extended information, existence of waiting task, etc.) for the mailbox specified by *mbxid* into the packet specified by *pk_rmbx*.

Mailbox information is described in detail below.

exinf ... Extended information

wtsk ... Existence of waiting task
FALSE (0): No waiting task
Value: ID number of the first task of wait queue

pk_msg ... Existence of waiting message
NADR (-1): No waiting message
Value: Address of the first message of wait queue

keyid ... Key ID number
FALSE (0): No key ID number specified at creation
Value: Key ID number

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store mailbox information is invalid (*pk_rmbx* = 0).

Macro	Value	Description
E_ID	-35	Invalid ID number specification (maximum number of mailboxes created < *mbxid*).
*E_NOEXS	-52	The target mailbox does not exist.
E_OACV	-66	An unauthorized ID number (*mbxid* $\leq$ 0) was specified.

vget_mid

get mailbox identifier (-245)

Task/Non-task

[Overview]

Acquires the mailbox ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = vget_mid ( ID *p_mbxid, ID keyid );
```

[Parameter(s)]

I/O	Parameter	Description
O	`ID *p_mbxid;`	Address of area used to store ID number
I	`ID keyid;`	Mailbox key ID number

[Explanation]

This system call stores the mailbox ID number specified by *keyid* in the area specified by *p_mbxid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the ID number is invalid (*p_mbxid* = 0). - Invalid key ID number specification (*keyid* = 0).
*E_NOEXS	-52	The target mailbox does not exist.

12.8.4 Interrupt management system calls

This section explains the group of system calls that perform processing that depends on maskable interrupts (interrupt management system calls).

Table 12-8 lists the interrupt management system calls.

Table 12-8 Interrupt Management System Calls

System Call	Function
def_int	Registers an indirectly activated interrupt handler and cancels its registration.
ena_int	Enables the acknowledgement of maskable interrupts.
dis_int	Disables the acknowledgement of maskable interrupts.
loc_cpu	Disables the acknowledgement of maskable interrupts and dispatch processing.
unl_cpu	Enables the acknowledgement of maskable interrupts and dispatch processing.
chg_icr	Changes the interrupt control register.
ref_icr	Acquires the interrupt control register.

def_int	define interrupt handler (-65) Task/Non-task

[Overview]

Registers an indirectly activated interrupt handler and cancels its registration.

[C format]

```
#include     <stdrx85p.h>
ER           ercd = def_int ( UINT eintno, T_DINT *pk_dint );
```

[Parameter(s)]

I/O	Parameter	Description
I	`UINT eintno;`	Interrupt source number of indirectly activated interrupt handler
I	`T_DINT *pk_dint;`	Start address of packet storing indirectly activated interrupt handler registration information

[Structure of indirectly activated interrupt handler registration information T_DINT]

```
typedef struct  t_dint {
  ATR   intatr;   /*Attribute of indirectly activated interrupt handler*/
  FP    inthdr;   /*Activation address of indirectly activated interrupt handler*/
  VP    gp;       /*gp register-specific value*/
  VP    tp;       /*tp register-specific value*/
} T_DINT;
```

[Explanation]

This system call uses the information specified by *pk_dint* to register the indirectly activated interrupt handler activated upon the occurrence of the maskable interrupt with the interrupt source number specified by *eintno*.

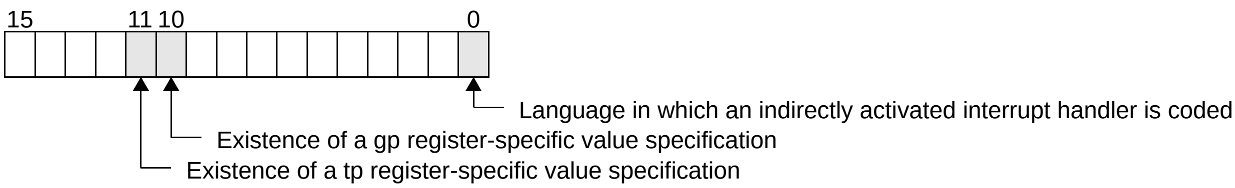
Indirectly activated interrupt handler registration information is described in detail below.

intatr ... Attribute of indirectly activated interrupt handler

Bit 0 ... Language in which an indirectly activated interrupt handler is coded
- TA_ASM (0): Assembly language
- TA_HLNG (1): C language

Bit 10 ... Existence of a gp register-specific value specification
- TA_DPID (1): gp register-specific value specified.

Bit 11 .. Existence of a tp register-specific value specification
- TA_DPIC (1): tp register-specific value specified.



inthdr ... Activation address of indirectly activated interrupt handler

gp ... gp register-specific value for indirectly activated interrupt handler

tp ... tp register-specific value for indirectly activated interrupt handler

When this system call is issued, if an indirectly activated interrupt handler corresponding to the specified interrupt source number has already been registered, this system call does not handle this as an error and newly registers the specified indirectly activated interrupt handler.

When this system call is issued, if NADR (-1) is set in the area specified by *pk_dint*, the registration of the interrupt handler specified by *eintno* is canceled.

Remark1 If the value of bit 10 is not 1 (TA_DPID), the contents of gp are meaningless.

Remark2 If the value of bit 11 is not 1 (TA_DPIC), the contents of tp are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	The interrupt handler management block area cannot be secured.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute intatr.
E_PAR	-33	Invalid parameter specification. - Invalid interrupt source number specification (*eintno* < 0, maximum interrupt source number < *eintno*). - The start address of the packet storing indirectly activated interrupt handler registration information is invalid (*pk_dint* = 0). - Invalid specification of the activation address (inthdr = 0).

ena_int

enable interrupt (-71)

Task/Non-task

[Overview]

Enables acknowledgement of maskable interrupts.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ena_int ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call allows the resumption of acknowledgement of maskable interrupts that were disabled by issuing dis_int.

If a maskable interrupt occurs after dis_int is issued before this system call is issued, the RX850 Pro delays switching to interrupt processing (a directly activated interrupt handler or an indirectly activated interrupt handler) until this system call is issued.

Remark This system call does not queue resume requests. Therefore, if this system call has been issued already and acknowledgement of maskable interrupts has been enabled, no processing is executed and it is not treated as an error.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.

dis_int

disable interrupt (-72)

Task/Non-task

[Overview]

Disables acknowledgement of maskable interrupts.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = dis_int ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call disables the acknowledgement of maskable interrupts.

This disables the acknowledgement of maskable interrupts before ena_int is issued.

If a maskable interrupt occurs after this system call is issued before ena_int is issued, the RX850 Pro delays switching to interrupt processing (a directly activated interrupt handler or an indirectly activated interrupt handler) until ena_int is issued.

Remark This system call does not queue disable requests. Therefore, if this system call has been issued already and acknowledgement of maskable interrupts has been disabled, no processing is executed and it is not treated as an error.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.

loc_cpu

lock CPU (-8)

Task

[Overview]

Disables the acknowledgement of maskable interrupts and dispatch processing.

[C format]

```
#include     <stdrx85p.h>
ER           ercd = loc_cpu ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call disables the acknowledgement of maskable interrupts and dispatch processing (task scheduling).

Therefore, for the period of time from the issuance of this system call to the issuance of unl_cpu, there is no transfer of control to another handler or task.

If a maskable interrupt occurs after this system call is issued but before unl_cpu is issued, the RX850 Pro delays processing for the interrupt (interrupt handler) until unl_cpu is issued. If a system call (chg_pri, sig_sem, etc.) requiring task scheduling is issued, the RX850 Pro merely queues the tasks into a wait queue and delays the actual scheduling until unl_cpu is issued, at which point all the tasks are processed in batch.

Remark This system call does not queue disable requests. Therefore, if this system call has been issued already and acknowledgement of maskable interrupts and dispatch processing has been disabled, no processing is executed and it is not treated as an error.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_CTX	-69	This system call was issued from a non-task.

unl_cpu

unlock CPU (-7)

Task

[Overview]

Enables the acknowledgement of maskable interrupts and dispatch processing.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = unl_cpu ( void );
```

[Parameter(s)]

None.

[Explanation]

This system call allows the resumption of acknowledgement of maskable interrupts and dispatch processing (task scheduling) disabled by issuing loc_cpu.

If a maskable interrupt occurs after loc_cpu is issued but before this system call is issued, the RX850 Pro delays processing for the interrupt (interrupt handler) until this system call is issued. If a system call (chg_pri, sig_sem, etc.) requiring task scheduling is issued, the RX850 Pro merely queues the tasks into a wait queue and delays actual scheduling until this system call is issued, at which point all the tasks are processed in batch.

Remark1 Dispatch processing that was disabled by the issuance of dis_dsp is reenabled by this system call.

Remark2 This system call does not queue resume requests. Therefore, if this system call has been issued already and acknowledgement of maskable interrupts and dispatch processing has resumed, no processing is executed and it is not treated as an error.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_CTX	-69	This system call was issued from a non-task.

chg_icr	change interrupt control register (-67) Task/Non-task

[Overview]

Changes the contents of the interrupt control register.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = chg_icr ( UINT eintno, UB icrcmd );
```

[Parameter(s)]

I/O	Parameter	Description
I	UINT *eintno*;	Interrupt source number
I	UB *icrcmd*;	Specification of interrupt request flag ICR_CLRINT (0x20): No interrupt request Specification of interrupt mask flag ICR_CLRMSK (0x10): Enables interrupt processing ICR_SETMSK (0x40): Disables interrupt processing Specification of changing interrupt priority order ICR_CHGLVL (0x08): Changes interrupt priority order Specification of interrupt priority order Value (0 to 7): Interrupt priority order

[Explanation]

This system call changes the contents of the interrupt control register specified by *eintno* to the value specified by *icrcmd*.

The *icrcmd* specification formats are as follows:

- *icrcmd* = ICR_CLRINT
  Changes the interrupt request flag of the interrupt control register to 0.

- *icrcmd* = ICR_CLRMSK
  Changes the interrupt mask flag of the interrupt control register to 0.

- *icrcmd* = ICR_SETMSK
  Changes the interrupt mask flag of the interrupt control register to 1.

- *icrcmd* = (ICR_CHGLVL | value)
  Changes the interrupt priority order of the interrupt control register to the value specified by "Value".
  The value "0" corresponds to level 0 and value "7" to level 7.

Remark1 Specify the value calculated by [(the exceptional code of the specified interrupt source number - 0x80) / 0x10] for the interrupt source number *eintno*.

Remark2 When the RX850 Pro is operated on the V850ES/V850E1/V850E2 core, even if this system call is issued, the desired interrupt control register may not operate. In the RX850 Pro, the interrupt control register address is calculated from the interrupt source number. However, in the V850ES/V850E1/V850E2 core, the correct register address cannot be obtained since the alignment of the interrupt source numbers and interrupt control registers differs from other V850 microcontrollers products. Therefore, use of this system call is restricted. For manipulating the interrupt control register via an application, directly manipulate the register without using this system call.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - Invalid specification of interrupt source number (*eintno* < 0, maximum interrupt source number < *eintno*). - Invalid specification of interrupt request flag (*eintno* < 0, maximum interrupt source number < *eintno*). - (ICR_CLRMSK || ICR_SETMSK) is specified as an interrupt request flag.

ref_icr	refer interrupt control register status (-68) Task/Non-task

[Overview]

Acquires the contents of the interrupt control register.

[C format]

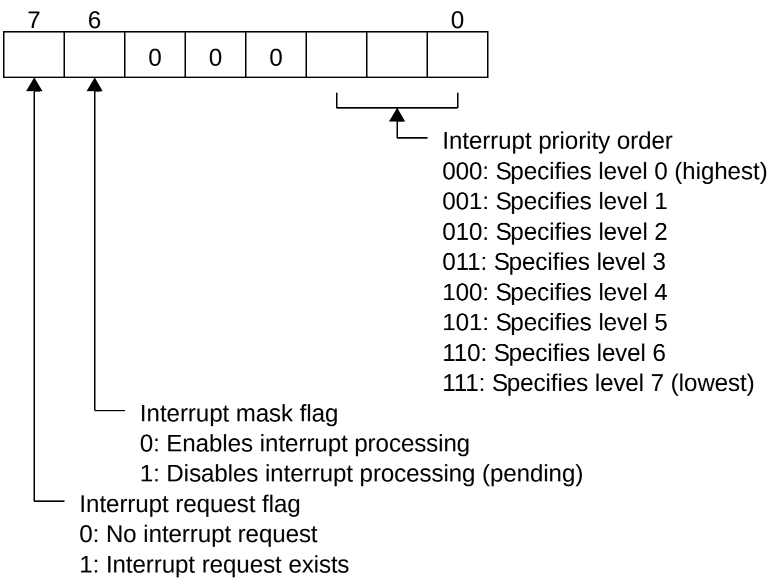
```
#include    <stdrx85p.h>
ER          ercd = ref_icr ( UB *p_regptn, UINT eintno );
```

[Parameter(s)]

I/O	Parameter	Description
O	`UB` *`*p_regptn`*`;`	Address of area used to store contents of interrupt control register
I	`UINT` *`eintno`*`;`	Interrupt source number Interrupt mask flag 0: Enables interrupt processing 1: Disables interrupt processing (pending) Interrupt request flag 0: No interrupt request 1: Interrupt request exists

[Explanation]

This system call stores the contents of the interrupt control register specified by *eintno* in the area specified by *p_regptn*. The following figure shows the contents of the acquired interrupt control register:



Remark1 Specify the value calculated by [(the exceptional code of the specified interrupt source number - 0x80) / 0x10] for the interrupt source number *eintno*.

Remark2 When the RX850 Pro is operated on the V850ES/V850E1/V850E2 core, even if this system call is issued, the desired interrupt control register may not operate. In the RX850 Pro, the interrupt control register address is calculated from the interrupt source number. However, in the V850ES/V850E1/V850E2 core, the correct register address cannot be obtained since the alignment of the interrupt source numbers and interrupt control registers differs from other V850 microcontrollers products. Therefore, use of this system call is restricted. For manipulating the interrupt control register via an application, directly manipulate the register without using this system call.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the contents of interrupt control register (*p_regptn*) is 0. - Invalid specification of interrupt source number (*eintno* < 0, maximum interrupt source number < *eintno*).

12.8.5 Memory pool management system calls

This section explains the group of system calls that allocate memory blocks (memory pool management system calls). Table 12-9 lists the memory pool management system calls.

Table 12-9 Memory Pool Management System Calls

System Call	Function
cre_mpl	Creates a memory pool.
del_mpl	Deletes a memory pool.
get_blk	Acquires a memory block.
pget_blk	Acquires a memory block (polling).
tget_blk	Acquires a memory block (with timeout).
rel_blk	Returns a memory block.
ref_mpl	Acquires memory pool information.
vget_pid	Acquires memory pool ID number.

cre_mpl

create variable-size memory pool (-137)

Task

[Overview]

Creates a memory pool.

[C format]

- When an ID number is specified

```
#include    <stdrx85p.h>
ER          ercd = cre_mpl ( ID mplid, T_CMPL *pk_cmpl );
```

- When an ID number is not specified

```
#include    <stdrx85p.h>
ER          ercd = cre_mpl ( ID_AUTO, T_CMPL *pk_cmpl, ID *p_mplid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID mplid;`	Memory pool ID number
I	`T_CMPL *pk_cmpl;`	Start address of packet containing memory pool creation information
O	`ID *p_mplid;`	Address of area used to store ID number

[Structure of memory pool creation information T_CMPL]

```
typedef struct  t_cmpl {
        VP      exinf;          /*Extended information*/
        ATR     mplatr;         /*Memory pool attribute*/
        INT     mplsz;          /*Memory pool size*/
        ID      keyid;          /*Memory pool key ID number*/
} T_CMPL;
```

[Explanation]

The RX850 Pro provides 2 types of interfaces for memory pool creation: one in which an ID number must be specified for memory pool creation, and one in which an ID number is not specified.

- When an ID number is specified
  A memory pool having the ID number specified by *mplid* is created based on the information specified by *pk_cmpl*.

- When an ID number is not specified
  A memory pool is created based on the information specified by *pk_cmpl*.
  An ID number is allocated by the RX850 Pro and the allocated ID number is stored in the area specified by *p_mplid*.

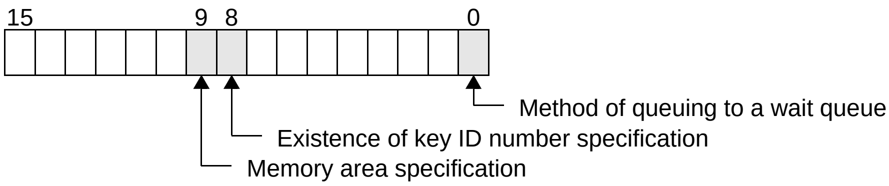
Memory pool creation information is described in detail below.

exinf ...   Extended information
exinf is an area used for storing user-specific information for the specified memory pool.
The user can use this area as required.
Information set in exinf can be dynamically acquired by issuing ref_mpl from a processing program (task/non-task).

mplatr ... Memory pool attribute
Bit 0 ... Method of queuing to a wait queue
TA_TPRI (0): Priority order
TA_TFIFO (1): FIFO order

Bit 8 ... Existence of key ID number specification
TA_KEYID (1): Key ID number specified

Bit 9 ... Memory area specification
TA_UPOL0 (0): Secures the memory pool area from user memory area
TA_UPOL1 (1): Secures the memory pool area from user memory area



mplsz ... Memory pool size (unit: byte)

keyid ... Memory pool key ID number

Remark If the value of bit 8 is not 1 (TA_KEYID) in mplatr, the contents of keyid are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOMEM	-10	A memory pool management block or memory pool area cannot be allocated.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute mplatr.
E_PAR	-33	Invalid parameter specification. - The start address of the packet storing memory pool creation information is invalid (*pk_cmpl* = 0). - Invalid size specification (*mplsz* ≤ 0). - Invalid key ID number specification (keyid = 0) (at TA_KEYID attribute specified). - The address of the area used to store the ID number is invalid (*p_mplid* = 0) (for creation without specifying the ID number).
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_OBJ	-63	A memory pool having the specified ID number has already been created.
E_OACV	-66	An unauthorized ID number (*mplid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

del_mpl

delete variable-size memory pool (-138)

Task

[Overview]

Deletes a memory pool.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = del_mpl ( ID mplid );
```

[Parameter(s)]

I/O	Parameter	Description
I	`ID mplid;`	Memory pool ID number

[Explanation]

This system call deletes the memory pool specified by *mplid*.

The target memory pool is released from the control of the RX850 Pro.

The task released from the wait state (memory block wait state) by this system call has E_DLT returned as the return value of the system call (get_blk or tget_blk) that instigated the transition to the wait state.

If this system call is issued when the task acquires a memory block that the target memory pool manages, the memory block is also released from the control of the RX850 Pro. Accordingly, the contents of the acquired memory block are undefined.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
E_OACV	-66	An unauthorized ID number (*mplid* ≤ 0) was specified.
E_CTX	-69	This system call was issued from a non-task.

get_blk

get variable-size memory block (-141)

Task

[Overview]

Acquires a memory block.

[C format]

```
#include      <stdrx85p.h>
ER            ercd = get_blk ( VP *p_blk, ID mplid, INT blksz );
```

[Parameter(s)]

I/O	Parameter	Description
O	VP *p_blk;	Address of area used to store start address of memory block
I	ID mplid;	Memory pool ID number
I	INT blksz;	Memory block size (unit: bytes)

[Explanation]

This system call acquires a memory block of the size specified by *blksz* from the memory pool specified by *mplid* and stores its start address in the area specified by *p_blk*.

If no memory block can be acquired from the target memory pool (when there is no free area of the requested size) upon the issuance of this system call, this system call places the task in the wait queue of the target memory pool before changing its state from the run state to the wait state (memory block wait state).

The memory block wait state is released when a memory block that satisfies the requested size is released by rel_blk or upon the issuance of del_mpl or rel_wai, and the task returns to the ready state.

Remark1 he RX850 Pro does not clear the memory upon acquiring a memory block. Accordingly, the contents of the acquired memory block are undefined.

Remark2 When a task queues in the wait queue of the target memory pool, it is executed in the order (FIFO order or priority order) specified when that memory pool was generated (at configuration or when cre_mpl was issued).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the start address of the memory block is invalid (*p_blk* = 0). - Invalid specification of memory block size (*blksz* ≤ 0).
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
E_OACV	-66	An unauthorized ID number (*mplid* ≤ 0) was specified.

Macro	Value	Description
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_DLT	-81	The target memory pool was deleted by del_mpl.
*E_RLWAI	-86	The memory block wait state was forcibly released by rel_wai.

pget_blk

poll and get variable-size memory block (-104)

Task/Non-task

[Overview]

Acquires a memory block (polling).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = pget_blk ( VP *p_blk, ID mplid, INT blksz );
```

[Parameter(s)]

I/O	Parameter	Description
O	`VP *p_blk;`	Address of area used to store start address of memory block
I	`ID mplid;`	Memory pool ID number
I	`INT blksz;`	Memory block size (unit: bytes)

[Explanation]

This system call acquires a memory block of the size specified by *blksz* from the memory pool specified by *mplid* and stores its start address in the area specified by *p_blk*.

When this system call is issued, if no memory block can be acquired from the target memory pool (when there is no free area of the requested size), this system call returns E_TMOUT as the return value.

Remark The RX850 Pro does not clear the memory upon acquiring a memory block. Accordingly, the contents of the acquired memory block are undefined.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the start address of a memory block is invalid (*p_blk* = 0). - Invalid specification of memory block size (*blksz* $\leq$ 0).
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
E_OACV	-66	An unauthorized ID number (*mplid* $\leq$ 0) was specified.
*E_TMOUT	-85	There is no free space in the target memory pool.

tget_blk	get variable-size memory block with timeout (-168) Task

[Overview]

Acquires a memory block (with timeout).

[C format]

```
#include    <stdrx85p.h>
ER          ercd = tget_blk ( VP *p_blk, ID mplid, INT blksz, TMO tmout );
```

[Parameter(s)]

I/O	Parameter	Description
O	`VP *p_blk;`	Address of area used to store start address of memory block
I	`ID mplid;`	Memory pool ID number
I	`INT blksz;`	Memory block size (unit: bytes)
I	`TMO tmout;`	Wait time (unit: ms) TMO_POL (0): Quick return TMO_FEVR (-1): Permanent wait Value: Wait time

[Explanation]

This system call acquires a memory block of the size specified by *blksz* from the memory pool specified by *mplid* and stores its start address in the area specified by *p_blk*.

If a memory block cannot be acquired from the target memory pool (when there is no free area of the requested size) when this system call is issued, this system call places the task in the wait queue of the target memory pool before changing it from the run state to the wait state (memory block wait state).

The memory block wait state is released when the wait time specified by *tmout* elapses, when a memory block that satisfies the requested size is released by rel_blk, or when del_mpl or rel_wai is issued. The task then returns to the ready state.

Remark1 The RX850 Pro does not clear the memory upon acquiring a memory block. Accordingly, the contents of the acquired memory block are undefined.

Remark2 When a task queues in the wait queue of the target memory pool, it is executed in the order (FIFO order or priority order) specified when that memory pool was generated (at configuration or when cre_mpl was issued).

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the start address of the memory block is invalid (*p_blk* = 0). - Invalid specification of memory block size (*blksz* ≤ 0).

Macro	Value	Description
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
E_OACV	-66	An unauthorized ID number (*mplid* $\leq$ 0) was specified.
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_DLT	-81	The target memory pool was deleted by del_mpl.
*E_TMOUT	-85	Timeout elapsed.
*E_RLWAI	-86	The memory block wait state was forcibly released by rel_wai.

rel_blk	release variable-size memory block (-143) Task/Non-task

[Overview]

Returns a memory block.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = rel_blk ( ID mplid, VP blk );
```

[Parameter(s)]

I/O	Parameter	Description
I	ID *mplid*;	Memory pool ID number
I	VP *blk*;	Start address of memory block

[Explanation]

This system call returns the memory block specified by *blk* to the memory pool specified by *mplid*.

If the size of the returned memory block satisfies the size requested by the (first) task queuing in the target memory pool's wait queue when this system call is issued, the memory block is transferred to that task.

The relevant task is consequently removed from the wait queue, and changes from the wait state (memory block wait state) to the ready state, or from the wait-suspend state to the suspend state.

Remark1 The contents of a returned memory block are not cleared by the RX850 Pro. Thus, the contents of a memory block may be undefined when that memory block is returned.

Remark2 The RX850 Pro includes 2 different specifications for this system call.

(1) When a memory block is returned by this system call, if the first 4 bytes of the memory block are not filled with zeros, the return value E_OBJ is used for termination instead of returning the memory block.

(2) When this system call is issued, the memory block is returned even if the first 4 bytes of the memory block are not filled with zeros (return value = E_OK).

The first specification applies when the memory block is used as a mailbox's message area, and this is the specification that has been used for this system call as it has been implemented thus far in the RX850 Pro.
When the memory block is used as a mailbox's message area, the first 4 bytes serve as the link area for the message's wait queue. In other words, if messages are queued in the mailbox, when this system call is issued and the memory block must be returned, in which case it is the message area linked to the queue that is returned. To prevent this, the specification requires the first 4 bytes that comprise the link area to be filled with zeros, otherwise it will be recognized as the memory block used as the message area and the return value E_OBJ will be used for termination instead of returning the memory block. Under this specification, the first 4 bytes must be cleared to zero in order to use this system call to return the memory block.
These specifications of this system call are stored in separate libraries so that one or the other this system call specification can be used. Link to the library of this system call specification to be used.

(1) Library containing this system call that requires zero-clearing of first 4 bytes of memory block
---> librxp.a

(2) Library containing this system call that does not require zero-clearing of first 4 bytes of memory block
---> librxpm.a

Remark3 Treat a memory pool that returns a memory block the same as a memory pool specified when issuing get_blk, pget_blk, or tget_blk.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
*E_PAR	-33	Invalid parameter specification. - Invalid specification of the start address of a memory block (*blk* = 0). - The memory pool specified when acquired differs from that specified upon the issuance of this system call.
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
*E_OBJ	-63	Invalid state of the specified memory block. - A value other than 0x0 is placed in the first 4 bytes of the memory block to be returned. - This return value is returned when returning the memory block used as a message area.
E_OACV	-66	An unauthorized ID number (*mplid* $\leq$ 0) was specified.

ref_mpl

refer variable-size memory pool status (-140)

Task/Non-task

[Overview]

Acquires memory pool information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_mpl ( T_RMPL *pk_rmpl, ID mplid );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_RMPL *pk_rmpl;	Start address of packet used to store memory pool information
I	ID mplid;	Memory pool ID number

[Structure of memory pool information T_RMPL]

```
typedef struct  t_rmpl {
        VP      exinf;          /*Extended information*/
        BOOL_ID wtsk;           /*Existence of waiting task*/
        INT     frsz;           /*Total size of free area*/
        INT     maxsz;          /*Maximum memory block size that can be acquired*/
        ID      keyid;          /*Key ID number*/
} T_RMPL;
```

[Explanation]

This system call stores the memory pool information (extended information, existence of waiting tasks, etc.) for the memory pool specified by *mplid* in the packet specified by *pk_rmpl*.

Memory pool information is described in detail below.

exinf ... Extended information

wtsk ... Existence of waiting task
FALSE (0): No waiting task
Value: ID number of first task in the wait queue

frsz ... Total size of free area (unit: bytes)

maxsz ... Maximum memory block size that can be acquired (unit: bytes)

keyid ... Key ID number
FALSE (0): No specification for key ID number at generation
Value: Key ID number

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store memory pool information is invalid (*pk_rmpl* = 0).

Macro	Value	Description
E_ID	-35	Invalid ID number specification (maximum number of memory pools created < *mplid*).
*E_NOEXS	-52	The target memory pool does not exist.
E_OACV	-66	An unauthorized ID number (*mplid* $\leq$ 0) was specified.

vget_pid

get variable-size memory pool identifier (-242)

Task/Non-task

[Overview]

Acquires the memory pool ID number.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = vget_pid ( ID *p_mplid, ID keyid );
```

[Parameter(s)]

I/O	Parameter	Description
O	ID *p_mplid;	Address of area used to store ID number
I	ID keyid;	Memory pool key ID number

[Explanation]

This system call stores the memory pool ID number specified by *keyid* in the area specified by *p_mplid*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The address of the area used to store the ID number is invalid (*p_mplid* = 0). - Invalid key ID number specification (*keyid* = 0).
*E_NOEXS	-52	The target memory pool does not exist.

12.8.6 Time management system calls

This section explains the group of system calls that perform processing dependent on time (time management system calls).

Table 12-10 lists the time management system calls.

Table 12-10 Time Management System Calls

System Call	Function
set_tim	Sets the system clock.
get_tim	Acquires the time from the system clock.
dly_tsk	Changes the task to the timeout wait state.
def_cyc	Registers a cyclic handler or cancels its registration.
act_cyc	Controls the activity state of a cyclic handler.
ref_cyc	Acquires cyclic handler information.

set_tim

set time (-83)

Task/Non-task

[Overview]

Sets the system clock.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = set_tim ( SYSTIME *pk_tim );
```

[Parameter(s)]

I/O	Parameter	Description
I	SYSTIME *pk_tim;	Start address of packet storing time

[Structure of system clock SYSTIME]

```
typedef struct  t_systime {
        UW      ltime;          /*Time (lower 32 bits)*/
        H       utime;          /*Time (higher 16 bits)*/
} SYSTIME;
```

[Explanation]

This system call sets the system clock to the time specified by *pk_tim*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet storing time is invalid (*pk_tim* = 0).

get_tim

get time (-84)

Task/Non-task

[Overview]

Acquires the time from the system clock.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = get_tim ( SYSTIME *pk_tim );
```

[Parameter(s)]

I/O	Parameter	Description
O	`SYSTIME *pk_tim;`	Start address of packet storing time

[Structure of system clock SYSTIME]

```
typedef struct  t_systime {
        UW      ltime;          /*Time (lower 32 bits)*/
        H       utime;          /*Time (higher 16 bits)*/
} SYSTIME;
```

[Explanation]

This system call sets the current system clock time in the packet specified by *pk_tim*.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet storing time is invalid (*pk_tim* = 0).

dly_tsk

delay task (-85)

Task

[Overview]

Changes the task to the timeout wait state.

[C format]

```
#include     <stdrx85p.h>
ER           ercd = dly_tsk ( DLYTIME dlytim );
```

[Parameter(s)]

I/O	Parameter	Description
I	`DLYTIME` *`dlytim`*`;`	Delay time (unit: ms)

[Explanation]

This system call changes the state of the task from the run state to the wait state (timeout wait state) for the delay time specified by *dlytim*.

The timeout wait state is released upon the elapse of the delay specified by *dlytim* or when rel_wai is issued. The task then returns to the ready state.

Remark The timeout wait state is released by neither wup_tsk.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid specification of delay time (*dlytim* < 0).
E_CTX	-69	Context error. - This system call was issued from a non-task. - This system call was issued in the dispatch disabled state.
*E_RLWAI	-86	The timeout wait state was forcibly released by rel_wai.

def_cyc

define cyclic handler (-90)

Task/Non-task

[Overview]

Registers a cyclic handler or cancels its registration.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = def_cyc ( HNO cycno, T_DCYC *pk_dcyc );
```

[Parameter(s)]

I/O	Parameter	Description
I	`HNNO cycno;`	Specification number of cyclic handler
I	`T_DCYC *pk_dcyc;`	Start address of packet storing cyclic handler registration information

[Structure of cyclic handler registration information T_DCYC]

```
typedef struct  t_dcyc {
    VP      exinf;  /*Extended information*/
    ATR     cycatr; /*Attribute of cyclic handler*/
    FP      cychdr; /*Activation address of cyclic handler*/
    UINT    cycact; /*Initial activity state of cyclic handler*/
    CYCTIME cyctim; /*Activation time interval of cyclic handler*/
    VP      gp;     /*gp register-specific value of cyclic handler*/
    VP      tp;     /*tp register-specific value of cyclic handler*/
} T_DCYC;
```

[Explanation]

This system call uses the information specified by *pk_dcyc* to register the cyclic handler having the specification number specified by *cycno*.

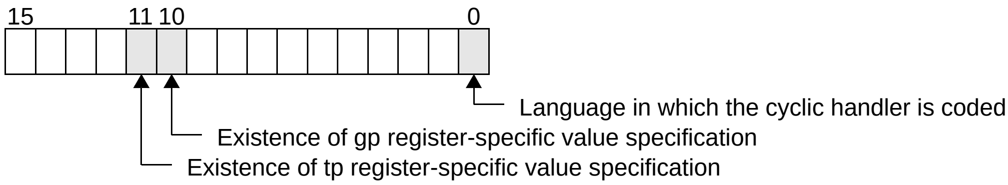
Cyclic handler registration information is described in detail below.

exinf ... Extended information
exinf is an area used for storing user-specific information on the specified cyclic handler. The user can use this area as required.
Information set in exinf can be dynamically acquired by issuing ref_cyc from a processing program (task/non-task).

cycatr ... Attribute of cyclic handler
Bit 0 ... Language in which the cyclic handler is coded
TA_ASM (0): Assembly language
TA_HLNG (1): C language

Bit 10 ... Existence of gp register-specific value specification
TA_DPID (1): gp register-specific value specified.

Bit 11 ... Existence of tp register-specific value specification
TA_DPIC (1): tp register-specific value specified.



cychdr ... Activation address of cyclic handler

cycact ... Initial activity state of cyclic handler
TCY_OFF (0): The initial activity state is OFF
TCY_ON (1): The initial activity state is ON

cyctim ... Activation time interval of cyclic handler (unit: basic clock cycles)

gp ... gp register-specific value for cyclic handler

tp ... tp register-specific value for cyclic handler

When this system call is issued, if a cyclic handler corresponding to the target specification number is already registered, this system call does not handle this as an error and newly registers the specified cyclic handler.

If this system call is issued with NADR (-1) set in the area specified by *pk_dcyc*, the registration of the cyclic handler specified by *cycno* is canceled.

Remark1 If the value of bit 10 is not 1 (TA_DPID) in cycatr, the contents of gp are meaningless.

Remark2 If the value of bit 11 is not 1 (TA_DPIC) in cycatr, the contents of tp are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute cycatr.
E_PAR	-33	Invalid parameter specification. - Invalid specification of specification number (cycno ≤ 0, maximum number of cyclic handlers that can be registered < *cycno*). - The start address of the packet storing cyclic handler registration information is invalid (*pk_dcyc* = 0). - Invalid specification of activation address (cychdr = 0). - Invalid specification of initial activity state cycact. - Invalid specification of activation time interval (cyctim ≤ 0).

act_cyc	activate cyclic handler (-94) Task/Non-task

[Overview]

Controls the activity state of a cyclic handler.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = act_cyc ( HNO cycno, UINT cycact );
```

[Parameter(s)]

I/O	Parameter	Description
I	HNO *cycno*;	Specification number of cyclic handler
I	UINT *cycact*;	Specification of activity state and cycle counter TCY_OFF (0): Changes the activity state to the OFF state. TCY_ON (1): Changes the activity state to the ON state. TCY_INI (2): Initializes the cycle counter.

[Explanation]

This system call changes the activity state of the cyclic handler specified by *cycno* to the state specified by *cycact*.
The specification format of *cycact* is described below.

- *cycact* = TCY_OFF
  Changes the activity state of the target cyclic handler to the OFF state.
  Even when the activation time is reached, the target cyclic handler is not activated.

  Remark Even when the activity state of the cyclic handler is OFF, the RX850 Pro increments the cycle counter.

- *cycact* = TCY_ON
  Changes the activity state of the target cyclic handler to the ON state.
  When the activation time is reached, the target cyclic handler is activated.

- *cycact* = TCY_INI
  Initializes the cycle counter of the target cyclic handler.

- *cycact* = (TCY_ON | TCY_INI)
  Changes the activity state of the target cyclic handler to the ON state before initializing the cycle counter.
  When the activation time is reached, the target cyclic handler is activated.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.

Macro	Value	Description
E_PAR	-33	Invalid parameter specification. - The specification number of the cyclic handler is invalid (*cycno* ≤ 0, maximum number of cyclic handlers that can be registered < *cycno*). - Invalid specification of activity state or cycle counter *cycact*.
*E_NOEXS	-52	The target cyclic handler is not registered.

ref_cyc

refer cyclic handler status (-92)

Task/Non-task

[Overview]

Acquires cyclic handler information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_cyc ( T_RCYC *pk_rcyc, HNO cycno );
```

[Parameter(s)]

I/O	Parameter	Description
O	`T_RCYC *pk_rcyc;`	Start address of packet used to store cyclic handler information
I	`HNO cycno;`	Specification number of cyclic handler

[Structure of cyclic handler information T_RCYC]

```
typedef struct  t_rcyc {
        VP      exinf;          /*Extended information*/
        CYCTIME lfttim;         /*Remaining time*/
        UINT    cycact;         /*Current activity state*/
} T_RCYC;
```

[Explanation]

This system call stores the cyclic handler information (extended information, remaining time, etc.) of the cyclic handler specified by *cycno* in the packet specified by *pk_rcyc*.

Cyclic handler information is described in detail below.

exinf ... Extended information

lfttim ... Time remaining until the cyclic handler is next activated (unit: basic clock cycles)

cycact ... Current activity state
TCY_OFF (0): Activity state is OFF.
TCY_ON (1): Activity state is ON.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	Invalid parameter specification. - The start address of the packet used to store cyclic handler information is invalid (*pk_rcyc* = 0). - The specification number of the cyclic handler is invalid (*cycno* ≤ 0, maximum number of cyclic handlers that can be registered < *cycno*).
*E_NOEXS	-52	The target cyclic handler is not registered.

12.8.7 System management system calls

This section explains the group of system calls that perform processing dependent on the system (system management system calls).

Table 12-11 lists the system management system calls.

Table 12-11 System Management System Calls

System Call	Function
get_ver	Acquires RX850 Pro version information.
ref_sys	Acquires system information.
def_svc	Registers an extended SVC handler or cancels its registration.
viss_svc	Calls an extended SVC handler.

get_ver

get version information (-16)

Task/Non-task

[Overview]

Acquires RX850 Pro version information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = get_ver ( T_VER *pk_ver );
```

[Parameter(s)]

I/O	Parameter	Description
O	T_VER *pk_ver;	Start address of packet used to store version information

[Structure of version information T_VER]

```
typedef struct  t_ver {
        UH      maker;      /*OS maker*/
        UH      id;         /*OS format*/
        UH      spver;      /*Specification version*/
        UH      prver;      /*OS version*/
        UH      prno[4];    /*Product number, production management information*/
        UH      cpu;        /*CPU information*/
        UH      var;        /*Variation descriptor*/
} T_VER;
```

[Explanation]

This system call stores the RX850 Pro version information (OS maker, OS format, etc.) in the packet specified by *pk_ver*.

Version information is described in detail below.

maker ... OS maker
0x0117: NEC Electronics Corporation

id ... OS format
0x0000: Not used

spver ... Specification version
0x5302: uITRON3.0 Ver.3.02

prver ... OS product version
0x0321: RX850 Pro Ver.3.21

prno[4] ... Product number/product management information
Undefined: Serial number of delivery product (each unit has a unique number)

cpu ... CPU information
0x0d33: V850 core
0x0d37: V850ES/V850E1/V850E2 core

var ... Variation descriptor
0xc000 uITRON level E, file not supported

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_PAR	-33	The start address of the packet used to store version information is invalid (*pk_ver* = 0).

ref_sys	refer system status (-12) Task/Non-task

[Overview]

Acquires system information.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = ref_sys ( T_RSYS *pk_rsys );
```

[Parameter(s)]

I/O	Parameter	Description
O	`T_RSYS *pk_rsys;`	Start address of packet used to store system information

[Structure of system information T_RSYS]

```
typedef struct  t_rsys {
        INT     sysstat;        /*System state*/
} T_RSYS;
```

[Explanation]

This system call stores the current value of dynamically-changing system information (system state) in the packet specified by *pk_rsys*.

System information is described in detail below.

sysstat ... System state

- TSS_TSK (0): Task processing is being performed. Dispatch processing is enabled.
- TSS_DDSP (1): Task processing is being performed. Dispatch processing is disabled.
- TSS_LOC (3): Task processing is being performed. The acknowledgement of maskable interrupts and dispatch processing is disabled.
- TSS_INDP (4): Processing of a non-task (interrupt handler, cyclic handler, etc.) is being performed.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
*E_PAR	-33	The start address of the packet used to store system information is invalid (*pk_rsys* = 0).

def_svc	define supervisor call handler (-9) Task/Non-task

[Overview]

Registers an extended SVC handler or cancels its registration.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = def_svc ( FN s_fncd, T_DSVC *pk_dsvc );
```

[Parameter(s)]

I/O	Parameter	Description
I	`FN` *`s_fncd;`*	Extended function code of extended SVC handler
I	`T_DSVC` *`*pk_dsvc;`*	Start address of packet storing the extended SVC handler registration

[Structure of extended SVC handler registration information T_DSVC]

```
typedef struct  t_dsvc {
        ATR     svcatr;   /*Attribute of extended SVC handler*/
        FP      svchdr;   /*Activation address of extended SVC handler*/
        VP      gp;       /*gp register-specific value for extended SVC handler*/
        VP      tp;       /*tp register-specific value for extended SVC handler*/
} T_DSVC;
```

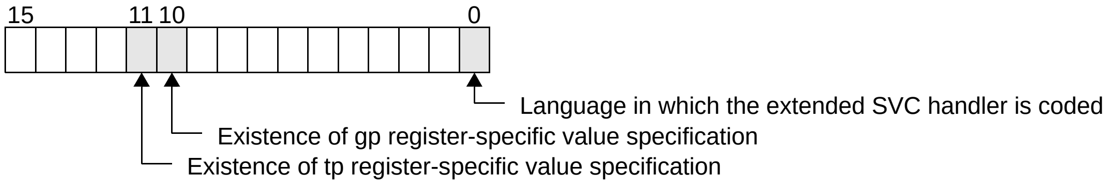
[Explanation]

This system call uses information specified by *pk_dsvc* to register the extended SVC handler having the extended function code specified by *s_fncd*.

Extended SVC handler registration information is described in detail below.

svcatr ... Attribute of extended SVC handler

- Bit 0 ... Language in which the extended SVC handler is coded
  - TA_ASM (0): Assembly language
  - TA_HLNG (1): C language
- Bit 10 ... Existence of go register-specific value specification
  - TA_DPID (1): gp register-specific value specified.
- Bit 11 ... Existence of tp register-specific value specification
  - TA_DPIC (1): tp register-specific value specified.



svchdr ... Activation address of extended SVC handler

gp ... gp register-specific value of extended SVC handler

tp ... tp register-specific value of extended SVC handler

When this system call is issued, if an extended SVC handler corresponding to the target extended function code has already been registered, this system call does not handle this as an error and newly registers the specified extended SVC handler.

When this system call is issued, if NADR (-1) is set in the area specified by *pk_dsvc*, the registration of the extended SVC handler specified by *s_fncd* is canceled.

Remark1 If the value of bit 10 is not 1 (TA_DPID) in svcatr, the contents of gp are meaningless.

Remark2 If the value of bit 11 is not 1 (TA_DPIC) in svcatr, the contents of tp are meaningless.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF.
E_RSATR	-24	Invalid specification of attribute svcatr.
E_PAR	-33	Invalid parameter specification. - Invalid specification of extended function code (*s_fncd* ≤ 0, maximum number of extended SVC handlers that can be registered < *s_fncd*). - The start address of the packet storing the extended SVC handler registration information is invalid (*pk_dsvc* = 0). - Invalid specification of activation address (svchdr = 0).

viss_svc

issued supervisor call handler (-250)

Task/Non-task

[Overview]

Calls an extended SVC handler.

[C format]

```
#include    <stdrx85p.h>
ER          ercd = viss_svc ( FN s_fncd, VW prm1, VW prm2, VW prm3 );
```

[Parameter(s)]

I/O	Parameter	Description
I	FN *s_fncd*;	Extended function code of extended SVC handler
I	VW *prm1*;	Parameter 1 passed to extended SVC handler
I	VW *prm2*;	Parameter 2 passed to extended SVC handler
I	VW *prm3*;	Parameter 3 passed to extended SVC handler

[Explanation]

This system call calls the extended SVC handler having the extended function code specified by *s_fncd*.

Remark When this system call is used to call an extended SVC handler, the interface library for the extended SVC handlers need not be coded.

[Return value]

Macro	Value	Description
*E_OK	0	Normal termination.
*E_NOSPT	-17	This system call is not defined as CF, or this system call calls an extended SVC handler that is not registered.
E_PAR	-33	Invalid specification of extended function code (*s_fncd* ≤ 0, maximum number of extended SVC handlers that can be registered < *s_fncd*).
Others	-	Return value from extended SVC handler.

CHAPTER 13 SYSTEM CONFIGURATION FILE

This chapter explains the system configuration file and how to describe it.

13.1 Outline

To organize a system using the RX850 Pro, various data (such as system information and resource information) and information containing the types of system calls to be used is necessary. The former is called the "system information table file" and the latter is called the "system call table file".

The system information table file and system call table file are described in assembly language and include data enumerated in a specified format. These tables can also be described using various editors, but it is time-consuming to describe them because changing and adding information is very difficult.

Therefore, an application called a configurator (CF850 Pro) is supplied.

This application converts a file in an original format, in which the system and resources of the RX850 Pro and information on the system calls to be used are described, into the system information table file and system call table file.

Therefore, the user can obtain the system information table file and system call table file by using the configurator and by creating a system configuration file.

The configurator outputs 3 files from the system configuration file: "system information table file", "system call table file", and "system information header file". The "system information header file" is a file in which symbol names specified as resource IDs, such as created tasks and semaphores, are made to correspond to the actual ID numbers by using the #define instruction.

For how to start the configurator, see "CHAPTER 14 CONFIGURATOR (CF850 Pro)".

How to describe the system configuration file is explained next.

13.2 Declaration

The following shows how to make the necessary declarations when describing a system configuration file.

1) Elements and character codes

a) Character codes
The system configuration file is described in ASCII code. Kanji character codes (SJIS and EUC only) can be used only in comments.

b) Words
In the system configuration file, any series of characters that does not contain any blank characters (space code, tab code, or line feed code) is called a word. In the following explanations, values, symbol names, and keywords are all words.
The configurator distinguishes between uppercase and lowercase characters. For example, "ABC," "Abc," and "abc" are handled as 3 different words.

c) Statements
In the system configuration file, a series of words delimited by one or more spaces is called a statement. One statement is delimited from another by a line feed.
In the system configuration file, a "\" appearing at the end of a line means that the line is continued on the next line. Note that a "\" must be preceded by a space or tab character.

2) Values
Any word beginning with a numerical code is treated as a value. Values are classified as shown in Table 13-1, according to the numerical code that appears at the beginning.
Unless otherwise specified, any 32-bit width (0x0 to 0xFFFFFFFF) can be specified.

Table 13-1 Types of Values

Type	Numerical Code at Beginning	Characters Used	Example
Decimal	0	0 to 7	0123, 0, 056712
Numerical	other than 0	0 to 9	123, 0, 689525

Type	Numerical Code at Beginning	Characters Used	Example
Octal	0x or 0X	0 to 9, a to f (, A to F), x, X	0x12C, 0X0, 0xabcdef

3) Symbol names
Symbol names are distinguished from names according to context. A symbol name indicates the name of a function or variable in a user program.
Note, however, that the first character of a symbol name must be an alphabetic character or an underline.

Remark Up to 255 characters can be used for specifying a symbol name.

4) Comments
Within a system configuration file, all text between "--" and the end of the line is handled as a comment.

5) Keywords
The character strings shown below are keywords reserved for use with the configurator.
The use of these character strings for any other purpose is prohibited.

clkhdr, clktim, cyc, defstk, flg, flgsvc, ini, inthdr, intstk, intsvc, maxcyc, maxflg, maxint, maxintfactor, maxmbx, maxmpl, maxpri, maxsem, maxsvc, maxtsk, mbx, mbxsvc, mem, mpl, mplsvc, no_use, prtflg, prtmbx, prtmpl, prtsem, prttsk, RX850PRO, rxsers, sct_def, sem, semsvc, ser_def, sit_def, SPOL0, SPOL1, svc, syssvc, TA_ASM, TA_DISINT, TA_ENAINT, TA_HLNG, TA_MFIFO, TA_MPRI, TA_TFIFO, TA_TPRI, TA_WMUL, TA_WSGL, TCY_OFF, TCY_ON, timsvc, tsk, tsksvc, TTS_DMT, TTS_RDY, UPOL0, UPOL1, V850, V850ES, V850E1, V850E2

13.3 Configuration Information

The system configuration information that is described in the system configuration file is divided into the following 3 main types.

- Real-time OS information
  Data related to the real-time OS being used.

- SIT information
  Data that is necessary for the operation of the RX850 Pro.

- SCT information
  Data that is used to select whether a system function (system call) is to be used.

13.3.1 Real-time OS information

The real-time OS information that is described in the system configuration file consists of the following 1 item.

1) RX series information
The following data is described as RX series information.

- Real-time OS name
- Version number

13.3.2 SIT information

The SIT information that is described in the system configuration file consists of the following 12 items.

1) System information
The following data is defined as system information.

- Processor type
- Basic clock cycle
- Clock handler number
- Default stack size
- Stack information for interrupt handler
  Stack size
  System memoy area type
- Range of protected ID numbers
  Range of protected task ID numbers
  Range of protected semaphore ID numbers
  Range of protected eventflag ID numbers
  Range of protected mailbox ID numbers
  Range of protected memory pool ID numbers

2) System maximum value information
The following data is defined as system maximum value information.

- Task priority range
- Maximum number of management objects that can be created or registered
  Maximum number of tasks
  Maximum number of semaphores
  Maximum number of eventflags
  Maximum number of mailboxes
  Maximum number of memory pools
  Maximum number of cyclic handlers
  Maximum number of extended SVC handlers
  Maximum number of indirectly activated interrupt handlers
  Maximum interrupt source number

3) System memory information
The following data is defined for each system memory area (System Memory Pool 0, System Memory Pool 1, User Memory Pool 0, User Memory Pool 1).

- Type
- Section name
- Size

4) Task information
The following data is defined for each task.

- ID number
- Initial status
- Activation code
- Extended information
- Description language
- Activation address
- Initial priority
- Interrupt mask status
- Stack information for task
  Stack size
  System memory area type
- gp register-specific value
- tp register-specific value

- Key ID number

5) Semaphore information
The following data is defined for each semaphore.

- ID number
- Extended information
- Task queuing method
- Initial resource count
- Maximum resource count
- Key ID number

6) Eventflag information
The following data is defined for each eventflag.

- ID number
- Extended information
- Whether waiting for multiple tasks
- Initial bit pattern
- Key ID number

7) Mailbox information
The following data can be defined for each mailbox.

- ID number
- Extended information
- Task queuing method
- Message queuing method
- Key ID number

8) Indirectly activated interrupt handler information
The following data is defined for each indirectly activated interrupt handler.

- Interrupt source number
- Description language
- Activation address
- gp register-specific value
- tp register-specific value

9) Memory pool information
The following data is defined for each memory pool.

- ID number
- Extended information
- Task queuing method
- Memory pool information
  memory pool size
  type of the system memory area to be allocated
- Key ID number

10) Cyclic handler information
The following data is defined for each cyclic handler.

- Specification number
- Extended information
- Description language
- Activation address
- Initial activation status

- Activation interval
- gp register-specific value
- tp register-specific value

11) Extended SVC handler information
The following data is defined for each extended SVC handler.

- Extended function code
- Description language
- Activation address
- gp register-specific value
- tp register-specific value

12) Initialization handler information
The following data is defined for each initialization handler.

- Description language
- Activation address
- gp register-specific value
- tp register-specific value

13.3.3 SCT information

The SCT information that is described in a system configuration file consists of the following 8 items.

1) Task management/task-associated synchronization management function system call information
Defines the task management/task-associated synchronization management function system calls used by a user program as the task management/task-associated synchronization management function system call information.
The task management/task-associated synchronization management function system calls supported by the RX850 Pro are listed below.

cre_tsk, del_tsk, sta_tsk, ext_tsk, exd_tsk, ter_tsk, dis_dsp, ena_dsp, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid, sus_tsk, rsm_tsk, frsm_tsk, slp_tsk, tslp_tsk, wup_tsk, can_wup

2) Synchronous communication (semaphore) management function system call information
Defines the semaphore management function system calls used by a user program as the semaphore management function system call information.
The semaphore management function system calls supported by the RX850 Pro are listed below.

cre_sem, del_sem, sig_sem, wai_sem, preq_sem, twai_sem, ref_sem, vget_sid

3) Synchronous communication (eventflag) management function system call information
Defines the eventflag management function system calls used by a user program as the eventflag management function system call information.
The eventflag management function system calls supported by the RX850 Pro are listed below.

cre_flg, del_flg, set_flg, clr_flg, wai_flg, pol_flg, twai_flg, ref_flg, vget_fid

4) Synchronous communication (mailbox) management function system call information
Defines the mailbox management function system calls used by a user program as the mailbox management function system call information.
The mailbox management function system calls supported by the RX850 Pro are listed below.

cre_mbx, del_mbx, snd_msg, rcv_msg, prcv_msg, trcv_msg, ref_mbx, vget_mid

5) Interrupt management function system call information
Defines the interrupt processing management function system calls used by a user program as the interrupt processing management function system call information.
The interrupt processing management function system calls supported by the RX850 Pro are listed below.

def_int, ena_int, dis_int, loc_cpu, unl_cpu, chg_icr, ref_icr

6) Memory pool management function system call information
Defines the memory pool management function system calls used by a user program as memory pool management function system call information.
The memory pool management function system calls supported by the RX850 Pro are listed below.

cre_mpl, del_mpl, get_blk, pget_blk, tget_blk, rel_blk, ref_mpl, vget_pid

7) Time management function system call information
Defines the time management function system calls used by a user program as the time management function system call information.
The time management function system calls supported by the RX850 Pro are listed below.

set_tim, get_tim, dly_tsk, def_cyc, act_cyc, ref_cyc

8) System management function system call information
Defines the system management function system calls used by a user program as the system management function system call information.
The system management function system calls supported by the RX850 Pro are listed below.

get_ver, ref_sys, def_svc, viss_svc

13.4 Specification Format for Real-Time OS Information

The following describes the format that must be observed when describing the real-time OS information in the system configuration file.

In the following explanation, bold text indicates a reserved word, while italics indicate a value, symbol name, or keyword to be supplied by the user.

13.4.1 RX series information

The RX series information defines values for the real-time OS name, version number.

For the system configuration file, the specification of RX series information is required.

The following shows the RX series information format.

```
rxsers      rtos_nam    rtos_ver
```

The items constituting the RX series information are as follows.

- *rtos_nam*
  Specifies the name of the real-time OS.
  The only keyword that can be specified for *rtos_nam* is “RX850PRO”.

- *rtos_ver*
  Specifies the version number of the real-time OS.
  The only keyword that can be specified for *rtos_ver* is “V32*x* (*x* is any number)”.

13.5 Specification Format for SIT Information

The following describes the format that must be observed when describing the SIT information in the system configuration file.

In the following explanation, bold text indicates a reserved word, while italics indicate a value, symbol name, or keyword to be supplied by the user.

13.5.1 System information

The system information defines values for the processor type, basic clock cycle, clock handler number, default stack size, stack information for interrupt handler, range of protected ID numbers.

For the system configuration file, the specification of the system information is required.

The following shows the system information format.

```
[ cputype       chip_type ]
clktim          time
clkhdr          clk_intno
defstk          stk_siz
intstk          intstk_siz:mem_nam
prttsk          tsk_idlmt
prtsem          sem_idlmt
prtflg          flg_idlmt
prtmbx          mbx_idlmt
prtmpl          mpl_idlmt
```

The items constituting the system information are as follows.

- *chip_type*
  Specifies the processor type of the target device.
  The keywords that can be specified for the processor type are V850ES, V850E1, and V850E2.

V850ES:	V850ES core
V850E1:	V850E1 core
V850E2:	V850E2 core

  If omitted: The processor type of target device is “V850E1”.

- *time*
  Specifies the basic clock cycle of the timer to be used (in ms).
  A value between 0x1 and 0x7fff can be specified for *time*.

  Remark1 The basic clock cycle is the cycle at which the RX850 Pro generates the clock interrupts necessary to realize the time management function (cycle rise, delay rise, timeout).

  Remark2 The timer that is used by the RX850 Pro for time management must be initialized so that an interrupt occurs in the 1-ms cycle.

- *clk_intno*
  Specifies a clock handler number.
  The values that can be specified as *clk_intno* is the interrupt source number specified with a device file, or a value calculated using "(exception code - 0x80) / 0x10".

- *stk_siz*
  Specifies the default stack size (in bytes).
  A value between 0x0 and 0x7ffffffc, aligned to a 4-byte boundary, can be specified for *stk_siz*.

  Remark The default stack size is the smallest task stack size that can exist within the system. If, therefore, at system activation, a static task is generated or if an active task is generated as a result of a cre_tsk, and a stack size smaller than the default is specified, that specification is ignored and the default size is used.

- *intstk_siz* : *mem_nam*
  Specifies the stack size to be used by a interrupt handler, and the type of the system memory area to be allocated to that stack (in bytes).
  A value between 0x0 and 0x7ffffffc, aligned to a 4-byte boundary, can be specified for *intstk_siz*.
  The keywords that can be specified for *mem_nam* are SPOL0 and SPOL1.

  SPOL0: Allocates the interrupt handler stack to System Memory Pool 0.
  SPOL1: Allocates the interrupt handler stack to System Memory Pool 1.

- *tsk_idlmt*
  Specifies the range of protected ID numbers when a task is generated with no ID number specified.
  A value between 0x0 and *tsk_cnt* can be specified for *tsk_idlmt*.

  Remark When 0x0 is specified for *tsk_idlmt*, no protection is imposed on the ID number. The value defined for the maximum number of tasks that can be created (maxtsk in the System maximum value information) is used for *tsk_cnt*.

- *sem_idlmt*
  Specifies the range of protected ID numbers when a semaphore is generated with no ID number specified.
  A value between 0x0 and *sem_cnt* can be specified for *sem_idlmt*.

  Remark When 0x0 is specified for *sem_idlmt*, no protection is imposed on the ID number. The value defined for the maximum number of semaphores that can be created (maxsem in the System maximum value information), is used for *sem_cnt*.

- *flg_idlmt*
  Specifies the range of protected ID numbers when an eventflag is generated with no ID number specified.
  A value between 0x0 and *flg_cnt* can be specified for *flg_idlmt*.

  Remark When 0x0 is specified for *flg_idlmt*, no protection is imposed on the ID number. The value defined for the maximum number of eventflags that can be created (maxflg in the System maximum value information), is used for *flg_cnt*.

- *mbx_idlmt*
  Specifies the range of protected ID numbers when a mailbox is generated with no ID number specified.
  A value between 0x0 and *mbx_cnt* can be specified for *mbx_idlmt*.

  Remark When 0x0 is specified for *mbx_idlmt*, no protection is imposed on the ID number. The value defined for the maximum number of mailboxes that can be created (maxmbx in the System maximum value information), is used for *mbx_cnt*.

- *mpl_idlmt*
  Specifies the range of protected ID numbers when a memory pool is generated with no ID number specified.
  A value between 0x0 and *mpl_cnt* can be specified for *mpl_idlmt*.

  Remark When 0x0 is specified for *mpl_idlmt*, no protection is imposed on the ID number. The value defined for the maximum number of memory pools that can be created (maxmpl in the System maximum value information), is used for *mpl_cnt*.

13.5.2 System maximum value information

The system maximum value information defines values for the task priority range, maximum number of management objects that can be created or registered.

For the system configuration file, the specification of the system maximum value information is required.

The following shows the system maximum value information format.

```
maxpri          pri_lvl
maxtsk          tsk_cnt
maxsem          sem_cnt
maxflg          flg_cnt
maxmbx          mbx_cnt
maxmpl          mpl_cnt
maxcyc          cyc_cnt
maxsvc          svc_cnt
maxint          ith_cnt
maxintfactor    itf_cnt
```

The items constituting the system maximum value information are as follows.

- *pri_lvl*
  Specifies the priority range (priority values) for the task.
  A value between 0x1 and 0xfc can be specified for *pri_lvl*.

- *tsk_cnt*
  Specifies the maximum number of tasks that can be created.
  A value between 0x1 and 0x7fff can be specified for *tsk_cnt*.

- *sem_cnt*
  Specifies the maximum number of semaphores that can be created.
  A value between 0x0 and 0x7fff can be specified for *sem_cnt*.

- *flg_cnt*
  Specifies the maximum number of eventflags that can be created.
  A value between 0x0 and 0x7fff can be specified for *flg_cnt*.

- *mbx_cnt*
  Specifies the maximum number of mailboxes that can be created.
  A value between 0x0 and 0x7fff can be specified for *mbx_cnt*.

- *mpl_cnt*
  Specifies the maximum number of memory pools that can be created.
  A value between 0x0 and 0x7fff can be specified for *mpl_cnt*.

- *cyc_cnt*
  Specifies the maximum number of cyclic handlers that can be registered.
  A value between 0x0 and 0x7fff can be specified for *cyc_cnt*.

- *svc_cnt*
  Specifies the maximum number of extended SVC handlers that can be registered.
  A value between 0x0 and 0x7fff can be specified for *svc_cnt*.

- *ith_cnt*
  Specifies the maximum number of interrupt handlers that can be registered.
  A value between 0x0 and (*itf_cnt* + 1) can be specified for *ith_cnt*.

- *itf_cnt*
  Specifies the maximum number of interrupt source specified in Indirectly activated interrupt handler information.
  The value that can be specified for itf_cnt is limited to the value obtained by "(max. value of interrupt exception code used by indirectly activated interrupt handler - 0x80) / 0x10" (0x0 to 0x7fff, or 0x0 to "max. value of interrupt exception code preset by the target processor - 0x80) / 0x10" when a device file is specified.

13.5.3 System memory information

The system memory information defines “the type, section name, and size of the system memory area” for each of the following memory blocks: System Memory Pool 0, System Memory Pool 1, User Memory Pool 0, and User Memory Pool 1.

For the system configuration file, the specification of the data for System Memory Pool 0 is required.

Also, for the system configuration file, when the data for User Memory Pool 1 is specified, the data for User Memory Pool 0 is required.

The following shows the system memory information format.

```
mem        mem_id        sec_nam        mem_siz
```

The items constituting the system memory information are as follows.

- *mem_id*
  Specifies the type of the system memory area.
  The keywords that can be specified for the system memory type are SPOL0, SPOL1, UPOL0, and UPOL1.

  SPOL0: System Memory Pool 0 is set as the system memory area type.
  SPOL1: System Memory Pool 1 is set as the system memory area type.
  UPOL0: User Memory Pool 0 is set as the system memory area type.
  UPOL1: User Memory Pool 1 is set as the system memory area type.

- *sec_nam*
  Specifies the section name of the memory area to which the system memory area is allocated.
  The values that can be specified as *sec_nam* are only the “section names (.*sec_nam*) defined in the link directive file, from which ‘.’ is deleted.”

  Remark For details on link directive files, refer to "2.6 Creating Llink Directive File".

- *mem_siz*
  Specifies the size of the system memory area (in bytes).
  A value between 0x100 and 0x7ffffffc, aligned to a 4-byte boundary, can be specified for *mem_siz*.

13.5.4 Task information

The task information defines the ID number, initial status, activation code, extended information, description language, activation address, initial priority, interrupt mask status, gp register-specific value, and tp register-specific value, key ID number for the task.

For the system configuration file, the specification of at least 1 item of task information is required.

The number of task information items that can be specified is defined as being within the range of 1 to the maximum number of tasks that can be registered, *tsk_cnt*, as set in the System maximum value information.

The following shows the task information format.

```
tsk         tsk_id      sts         sta_code    ext_inf     lang        \
            sta_adr     pri         intr        stk_siz:mem_nam         \
            data        text        key_id
```

The items constituting the task information are as follows.

- *tsk_id*
  Specifies the ID number of the task.
  A value between 0x0 and *tsk_cnt*, or a symbol name, can be specified for *tsk_id*.
  When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *tsk_idlmt* and *tsk_cnt*.
  The value defined for the task ID number protected range (prttsk in the System information) is set as *tsk_idlmt*.
  The value defined fot the maximum number of tasks that can be registered (maxtsk in the System maximum value information) is set as *tsk_cnt*.

- *sts*
  Specifies the initial status of the task.
  The keywords that can be specified for *sts* are TTS_DMT and TTS_RDY.

TTS_DMT:	The system enters the dormant status upon being activated.
TTS_RDY:	The system enters the ready status upon being activated.

  Remark If the initial status of every static task is set to TTS_DMT, it is assumed that there are no active tasks when the system is activated. In this case, an appropriate task must be activated using the initialization handler.

- *sta_code*
  Specifies the activation code of the task.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *sta_code*.

  Remark *sta_code* is valid only when TTS_RDY is specified for sts. It is invalid when TTS_DMT is specified for *sts*.

- *ext_inf*
  Specifies the extended information of the task.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant task. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_tsk by a processing program (task/non-task).

- *lang*
  Specifies the language used to describe the task.
  The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

TA_HLNG:	A task is described in C language.
TA_ASM:	A task is described in assembly language.

- *sta_adr*
  Specifies the activation address of the task.
  A value between 0x0 and 0xfffffffe, aligned to a 2-byte boundary, or a symbol name, can be specified for *sta_adr*.

- *pri*
  Specifies the initial priority of the task.
  A value between 0x1 and *pri_lvl* can be specified for *pri*.
  The value defined for the task priority range (maxpri in the System maximum value information) is set as *pri_lvl*.

- *intr*
  Specifies the interrupt status at task activation.
  The keywords that can be specified for *intr* are TA_ENAINT and TA_DISINT.

  TA_ENAINT: All interrupts are enabled at task activation.
  TA_DISINT: All interrupts are disabled at task activation.

- *stk_siz* : *mem_nam*
  Specifies the stack size to be used by a task, and the type of the system memory area to be allocated to that stack (in bytes).
  A value between 0x0 and 0x7ffffffc, aligned to a 4-byte boundary, can be specified for *stk_siz*.
  The keywords that can be specified for *mem_nam* are SPOL0 and SPOL1.

  SPOL0: Allocates the task stack to System Memory Pool 0.
  SPOL1: Allocates the task stack to System Memory Pool 1.

- *data*
  Specifies the gp register-specific value of the task.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* and no_use can be specified as a keyword.

  no_use: A gp register-specific value is not set.

- *text*
  Specifies the tp register-specific value of the task.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* and no_use can be specified as a keyword.

  no_use: A tp register-specific value is not set.

- *key_id*
  Specifies the key ID number of the task.
  A value between 0x0 and 0x7fff can be specified for *key_id*.

  Remark When 0x0 is specified for *key_id*, the configurator does not assign a key ID number.

13.5.5 Semaphore information

The semaphore information defines the ID number, extended information, task queuing method, initial resource count, maximum resource count, key ID number for the semaphore.

The number of semaphore information items that can be specified is defined as being within the range of 0 to the maximum number of semaphores that can be registered, *sem_cnt*, as set in the System maximum value information.

The following shows the semaphore information format.

```
sem        sem_id       ext_inf      twai_opt     init_cnt     max_cnt      key_id
```

The items constituting the semaphore information are as follows.

- *sem_id*
  Specifies the ID number of the semaphore.
  A value between 0x0 and *sem_cnt*, or a symbol name, can be specified for *sem_id*.
  When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *sem_idlmt* and *sem_cnt*.
  The value defined for the semaphore ID number protected range (prtsem in the System information) is set as *sem_idlmt*.
  The value defined fot the maximum number of semaphores that can be registered (maxsem in the System maximum value information) is set as *sem_cnt*.

- *ext_inf*
  Specifies the extended information of the semaphore.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant semaphore. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_sem by a processing program (task/non-task).

- *twai_opt*
  Specifies the task queuing method.
  The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

  TA_TFIFO: Tasks are queued in the same order as that in which resource requests are issued.
  TA_TPRI: Tasks are queued according to their priority.

- *init_cnt*
  Specifies the initial resource count of the semaphore.
  A value between 0x0 and *max_cnt* can be specified for *init_cnt*.

- *max_cnt*
  Specifies the maximum resource count of the semaphore.
  A value between 0x1 and 0x7fffffff can be specified for *max_cnt*.

- *key_id*
  Specifies the key ID number of the semaphore.
  A value between 0x0 and 0x7fff can be specified for *key_id*.

  Remark When 0x0 is specified for *key_id*, the configurator does not assign a key ID number.

13.5.6 Eventflag information

The eventflag information defines the ID number, extended information, whether waiting for multiple tasks, initial bit pattern, key ID number for the eventflag.

The number of ceventflag information items that can be specified is defined as being within the range of 0 to the maximum number of eventflags that can be registered, *flg_cnt*, as set in the System maximum value information.

The following shows the eventflag information format.

```
flg          flg_id       ext_inf      twai_opt     init_ptn     key_id
```

The items constituting the eventflag information are as follows.

- *fig_id*
  Specifies the ID number of the eventflag.
  A value between 0x0 and *flg_cnt*, or a symbol name, can be specified for *flg_id*.
  When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *flg_idlmt* and *flg_cnt*.
  The value defined for the eventflag ID number protected range (prtflg in the System information) is set as *flg_idlmt*.
  The value defined fot the maximum number of eventflags that can be registered (maxflg in the System maximum value information) is set as *flg_cnt*.

- *ext_inf*
  Specifies the extended information of the eventflag.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant eventflag. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_flg by a processing program (task/non-task).

- *twai_opt*
  Specifies whether wait for multiple tasks is disabled/enabled.
  The keywords that can be specified for *twai_opt* are TA_WSGL and TA_WMUL.

  TA_WSGL: Wait for multiple tasks is disabled.
  TA_WMUL: Wait for multiple tasks is enabled.

- *init_ptn*
  Specifies the initial bit pattern (32-bit width) of the eventflag.
  A value between 0x0 and 0xffffffff can be specified for *init_ptn*.

- *key_id*
  Specifies the key ID number. of the eventflag
  A value between 0x0 and 0x7fff can be specified for *key_id*.

  Remark When 0x0 is specified for *key_id*, the configurator does not assign a key ID number.

13.5.7 Mailbox information

The mailbox information defines the ID number, extended information, task queuing method, message queuing method, key ID number for the mailbox.

The number of mailbox information items that can be specified is defined as being within the range of 0 to the maximum number of maiboxes that can be registered, *mbx_cnt*, as set in the System maximum value information.

The following shows the mailbox information format.

```
mbx          mbx_id       ext_inf      twai_opt     mwai_opt     key_id
```

The items constituting the mailbox information are as follows.

- *mbx_id*
  Specifies the ID number of the mailbox.
  A value between 0x0 and *mbx_cnt*, or a symbol name, can be specified for *mbx_id*.
  When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *mbx_idlmt* and *mbx_cnt*.
  The value defined for the mailbox ID number protected range (prtmbx in the System information) is set as *mbx_idlmt*.
  The value defined fot the maximum number of mailboxes that can be registered (maxmbx in the System maximum value information) is set as *mbx_cnt*.

- *ext_inf*
  Specifies the extended information of the mailbox.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant mailbox. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_mbx by a processing program (task/non-task).

- *twai_opt*
  Specifies the task queuing method.
  The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

  TA_TFIFO: Tasks are queued in the same order as that in which message receive requests are issued.
  TA_TPRI: Tasks are queued according to their priority.

- *mwai_opt*
  Specifies the message queuing method.
  The keywords that can be specified for *mwai_opt* are TA_MFIFO and TA_MPRI.

  TA_MFIFO: Messages are queued in the same order as that in which messages are issued.
  TA_MPRI: Messages are queued according to their priority.

- *key_id*
  Specifies the key ID number of the mailbox.
  A value between 0x0 and 0x7fff can be specified for *key_id*.

  Remark When 0x0 is specified for *key_id*, the configurator does not assign a key ID number.

13.5.8 Indirectly activated interrupt handler information

The indirectly activated interrupt handler information defines the interrupt source number, description language, activation address, gp register-specific value, and tp register-specific value for the indirectly activated interrupt handler.

The number of indirectly activated interrupt handler information items that can be specified is defined as being within the range of 0 to the maximum number of indirectly activated interrupt handlers that can be registered, *ith_cnt*, as set in the System maximum value information.

The following shows the indirectly activated interrupt handler information format.

```
inthdr      int_no      lang        hdr_adr     data        text
```

The items constituting the indirectly activated interrupt handler information are as follows.

- *int_no*
  Specifies the interrupt source number of the indirectly activated interrupt handler.
  The values that can be specified as *int_no* is the interrupt source number specified with a device file, or a value calculated using "(exception code - 0x80) / 0x10".

  Remark The same interrupt source number cannot be specified for *int_no* and *clk_intno*.
  clk_intno is a value defined in a clock handler number clkhdr of System information.

- *lang*
  Specifies the language used to describe the indirectly activated interrupt handler.
  The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

  TA_HLNG: A indirectly activated interrupt handler is described in C language.
  TA_ASM: A indirectly activated interrupt handler is described in assembly language.

- *hdr_adr*
  Specifies the activation address of the indirectly activated interrupt handler.
  A value between 0x0 and 0xfffffffe, aligned to a 2-byte boundary, or a symbol name, can be specified for *hdr_adr*.

- *data*
  Specifies the gp register-specific value of the indirectly activated interrupt handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* and no_use can be specified as a keyword.

  no_use: A gp register-specific value is not set.

- *text*
  Specifies the tp register-specific value of the indirectly activated interrupt handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* and no_use can be specified as a keyword.

  no_use: A tp register-specific value is not set.

13.5.9 Memory pool information

The memory pool information defines the ID number, extended information, task queuing method, memory pool information, key ID number for the memory pool.

The number of memory pool information items that can be specified is defined as being within the range of 0 to the maximum number of memory pools that can be registered, *mpl_cnt*, as set in the System maximum value information.

The following shows the memory pool information format.

```
mpl          mpl_id       ext_inf      twai_opt     mpl_siz:mem_nam          key_id
```

The items constituting the memory pool information are as follows.

- *mpl_id*
  Specifies the ID number of the memory pool.
  A value between 0x0 and *mpl_cnt*, or a symbol name, can be specified for *mpl_id*.
  When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *mpl_idlmt* and *mpl_cnt*.
  The value defined for the memory pool ID number protected range (prtmpl in the System information) is set as *mpl_idlmt*.
  The value defined fot the maximum number of memory pools that can be registered (maxmpl in the System maximum value information) is set as *mpl_cnt*.

- *ext_inf*
  Specifies the extended information of the memory pool.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant memory pool. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_mpl by a processing program (task/non-task).

- *twai_opt*
  Specifies the task queuing method.
  The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

  TA_TFIFO: Tasks are queued in the same order as that in which memory block requests are issued.
  TA_TPRI: Tasks are queued according to their priority.

- *mpl_siz* : *mem_nam*
  Specifies the memory pool size, and the type of the system memory area to be allocated to that memory pool (in bytes).
  A value between 0x4 and 0x7ffffffc, aligned to a 4-byte boundary, can be specified for *mpl_siz*.
  The keywords that can be specified for *mem_nam* are UPOL0 and UPOL1.

  UPOL0: Allocates the memory pool to User Memory Pool 0.
  UPOL1: Allocates the memory pool to User Memory Pool 1.

- *key_id*
  Specifies the key ID number of the memory pool.
  A value between 0x0 and 0x7fff can be specified for *key_id*.

  Remark When 0x0 is specified for *key_id*, the configurator does not assign a key ID number.

13.5.10 Cyclic handler information

The cyclic handler information defines the specification number, extended information, description language, activation address, initial activation status, activation interval, gp register-specific value, and tp register-specific value for the cyclic handler.

The number of cyclic handler information items that can be specified is defined as being within the range of 0 to the maximum number of cyclic handlers that can be registered, *cyc_cnt*, as set in the System maximum value information.

The following shows the cyclic handler information format.

```
cyc          cyc_no      ext_inf     lang        hdr_adr     act         \
             intvl       data        text
```

The items constituting the cyclic handler information are as follows.

- *cyc_no*
  Specifies the specification number of the cyclic handler.
  A value between 0x1 and *cyc_cnt*, or a symbol name, can be specified for *cyc_no*.
  When a symbol name is specified, the configurator automatically assigns an unused ID number between 0x1 and *cyc_cnt*.
  The value defined fot the maximum number of cyclic handlers that can be registered (maxcyc in the System maximum value information) is set as *cyc_cnt*.

- *ext_inf*
  Specifies the extended information of the cyclic handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

  Remark *ext_inf* is provided to enable the specification of user own information for the relevant cyclic handler. The user can specify it as necessary.
  The value specified for *ext_inf* can be dynamically allocated upon the issuance of ref_cyc by a processing program (task/non-task).

- *lang*
  Specifies the language used to describe the cyclic handler.
  The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

  TA_HLNG: A cyclic handler is described in C language.
  TA_ASM: A cyclic handler is described in assembly language.

- *hdr_adr*
  Specifies the activation address of the cyclic handler.
  A value between 0x0 and 0xfffffffe, aligned to a 2-byte boundary, or a symbol name, can be specified for *hdr_adr*.

- *act*
  Specifies the initial activation status of the cyclic handler.
  The keywords that can be specified for *act* are TCY_ON and TCY_OFF.

  TCY_ON: The system enters the ON status upon being activated.
  TCY_OFF: The system enters the OFF status upon being activated.

- *intvl*
  Specifies the activation interval of the cyclic handler (in basic clock cycle).
  A value between 0x1 and 0xffffffff can be specified for *intvl*.

- *data*
  Specifies the gp register-specific value of the cyclic handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* and no_use can be specified as a keyword.

  no_use: A gp register-specific value is not set.

- *text*
  Specifies the tp register-specific value of the cyclic handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* and no_use can be specified as a keyword.

  no_use: A tp register-specific value is not set.

13.5.11 Extended SVC handler information

The extended SVC handler information defines the extended function code, description language, activation address, gp register-specific value, and tp register-specific value for the extended SVC handler.

The number of extended SVC handler information items that can be specified is defined as being within the range of 0 to the maximum number of extended SVC handlers that can be registered, *svc_cnt*, as set in the System maximum value information.

The following shows the extended SVC handler information format.

```
svc          svc_no       lang         hdr_adr      data         text
```

The items constituting the extended SVC handler information are as follows.

- *svc_no*
  Specifies the extended function code of the extended SVC handler.
  A value between 0x1 and *svc_cnt*, or a symbol name, can be specified for *svc_no*.
  When a symbol name is specified, the configurator automatically assigns an unused ID number between 0x1 and *svc_cnt*.
  The value defined fot the maximum number of extended SVC handlers that can be registered (maxsvc in the System maximum value information) is set as svc_cnt.

- *lang*
  Specifies the language used to describe the extended SVC handler.
  The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

TA_HLNG:	A extended SVC handler is described in C language.
TA_ASM:	A extended SVC handler is described in assembly language.

- *hdr_adr*
  Specifies the activation address of the extended SVC handler.
  A value between 0x0 and 0xfffffffe, aligned to a 2-byte boundary, or a symbol name, can be specified for *hdr_adr*.

- *data*
  Specifies the gp register-specific value of the extended SVC handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* and no_use can be specified as a keyword.

  no_use: A gp register-specific value is not set.

- *text*
  Specifies the tp register-specific value of the extended SVC handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* and no_use can be specified as a keyword.

  no_use: A tp register-specific value is not set.

13.5.12 Initialization handler information

The initialization handler information defines the description language, activation address, gp register-specific value, and tp register-specific value for the initialization handler.

Information of the initialization handler can be omitted in the system configuration file. If it is omitted, the RX850 Pro assumes that there is no initialization handler, and continues processing.

The following shows the initialization handler information format.

```
ini        lang        hdr_adr     data        text
```

The items constituting the initialization handler information are as follows.

- *lang*
  Specifies the language used to describe the initialization handler.
  The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

  TA_HLNG: A initialization handler is described in C language.
  TA_ASM: A initialization handler is described in assembly language.

- *hdr_adr*
  Specifies the activation address of the initialization handler.
  A value between 0x0 and 0xfffffffe, aligned to a 2-byte boundary, or a symbol name, can be specified for *hdr_adr*.

- *data*
  Specifies the gp register-specific value of the initialization handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* and no_use can be specified as a keyword.

  no_use: A gp register-specific value is not set.

- *text*
  Specifies the tp register-specific value of the initialization handler.
  A value between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* and no_use can be specified as a keyword.

  no_use: A tp register-specific value is not set.

13.6 Specification Format for SCT Information

The following describes the format that must be observed when describing the SCT information in the system configuration file.

In the following explanation, bold text indicates a reserved word, while italics indicate a value, symbol name, or keyword to be supplied by the user.

13.6.1 Task management/task-associated synchronization management function system call information

The task management/task-associated synchronization management function system call information defines data that indicates the task management/task-associated synchronization management function system calls used by a user processing program for each system call.

If the task management/task-associated synchronization management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the task management/task-associated synchronization management function system call information format.

```
tksvc      svc_nam
```

The items constituting the task management/task-associated synchronization management function system call information are as follows.

- *svc_nam*
  Specifies a task management/task-associated synchronization management function system call name.
  The following keywords can be specified for *svc_nam*.

  cre_tsk, del_tsk, sta_tsk, ext_tsk, exd_tsk, ter_tsk, dis_dsp, ena_dsp, chg_pri, rot_rdq, rel_wai, get_tid, ref_tsk, vget_tid, sus_tsk, rsm_tsk, frsm_tsk, slp_tsk, tslp_tsk, wup_tsk, can_wup

13.6.2 Synchronous communication (semaphore) management function system call information

The synchronous communication (semaphore) management function system call information defines data that indicates the synchronous communication (semaphore) management function system calls used by a user processing program for each system call.

If the synchronous communication (semaphore) management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the synchronous communication (semaphore) management function system call information format.

```
semsvc      svc_nam
```

The items constituting the synchronous communication (semaphore) management function system call information are as follows.

- *svc_nam*
  Specifies a synchronous communication (semaphore) management function system call name.
  The following keywords can be specified for *svc_nam*.

  cre_sem, del_sem, sig_sem, wai_sem, preq_sem, twai_sem, ref_sem, vget_sid

13.6.3 Synchronous communication (eventflag) management function system call information

The synchronous communication (eventflag) management function system call information defines data that indicates the synchronous communication (eventflag) management function system calls used by a user processing program for each system call.

If the synchronous communication (eventflag) management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the synchronous communication (evet flag) management function system call information format.

```
flgsvc      svc_nam
```

The items constituting the synchronous communication (eventflag) management function system call information are as follows.

- *svc_nam*
  Specifies a synchronous communication (eventflag) management function system call name.
  The following keywords can be specified for *svc_nam*.

  cre_flg, del_flg, set_flg, clr_flg, wai_flg, pol_flg, twai_flg, ref_flg, vget_fid

13.6.4 Synchronous communication (mailbox) management function system call information

The synchronous communication (mailbox) management function system call information defines data that indicates the synchronous communication (mailbox) management function system calls used by a user processing program for each system call.

If the synchronous communication (mailbox) management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the synchronous communication (mailbox) management function system call information format.

```
mbxsvc      svc_nam
```

The items constituting the synchronous communication (mailbox) management function system call information are as follows.

- *svc_nam*
  Specifies a synchronous communication (mailbox) management function system call name.
  The following keywords can be specified for *svc_nam*.

  cre_mbx, del_mbx, snd_msg, rcv_msg, prcv_msg, trcv_msg, ref_mbx, vget_mid

13.6.5 Interrupt management function system call information

The interrupt management function system call information defines data that indicates the interrupt management function system calls used by a user processing program for each system call.

If the interrupt management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the interrupt management function system call information format.

```
intsvc      svc_nam
```

The items constituting the interrupt management function system call information are as follows.

- *svc_nam*
  Specifies a interrupt management function system call name.
  The following keywords can be specified for *svc_nam*.

  def_int, ena_int, dis_int, loc_cpu, unl_cpu, chg_icr, ref_icr

13.6.6 Memory pool management function system call information

The memory pool management function system call information defines data that indicates the memory pool management function system calls used by a user processing program for each system call.

If the memory pool management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the memory pool management function system call information format.

```
mplsvc      svc_nam
```

The items constituting the memory pool management function system call information are as follows.

- *svc_nam*
  Specifies a memory pool management function system call name.
  The following keywords can be specified for *svc_nam*.

    cre_mpl, del_mpl, get_blk, pget_blk, tget_blk, rel_blk, ref_mpl, vget_pid

13.6.7 Time management function system call information

The time management function system call information defines data that indicates the time management function system calls used by a user processing program for each system call.

If the time management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the time management function system call information format.

```
timsvc      svc_nam
```

The items constituting the time management function system call information are as follows.

- *svc_nam*
  Specifies a time management function system call name.
  The following keywords can be specified for *svc_nam*.

  set_tim, get_tim, dly_tsk, def_cyc, act_cyc, ref_cyc

13.6.8 System management function system call information

The system management function system call information defines data that indicates the system management function system calls used by a user processing program for each system call.

If the system management function system call information is not defined and system call is used in an application, E_NOSPT (-17) is returned as the return value of the system call.

The following shows the system management function system call information format.

```
syssvc      svc_nam
```

The items constituting the system management function system call information are as follows.

- *svc_nam*
  Specifies a system management function system call name.
  The following keywords can be specified for *svc_nam*.

  get_ver, ref_sys, def_svc, viss_svc

13.7 Cautions

In the system configuration file, describe the system configuration information (real-time OS information, SIT information, SCT information) in the following order.

1) Declaration of the start of the Real-time OS information description

2) Real-time OS information description

3) Declaration of the start of the SIT information description

4) SIT information description

5) Declaration of the start of the SCT information description

6) SCT information description

Figure 13-1 illustrates how the system configuration file is described.

Figure 13-1 System Configuration File Description Format

```
-- Declaration of the start of the Real-time OS information description
ser_def

-- Real-time OS information description
:
:
:

-- Declaration of the start of the SIT information description
sit_def

-- SIT information description
:
:
:

-- Declaration of the start of the SCT information description
sct_def

-- SCT information description
:
:
:
```

13.8 Description Example

The following describes an example for coding the system configuration file.
The data items shown below are written in the coding example.

< Real-time OS information >

1) RX series information

Real-time OS name:	RX850PRO
Version number:	V321

< SIT information >

1) System information

Processor type:	V850E1 core
Basic clock cycle:	0x1 ms
Clock handler number:	0x1c (INTCMD0)
Default stack size:	0x100 bytes
Stack information for interrupt handler:	Allocates a memory area for 0x100 bytes, starting from System Memory Pool 0
Range of protected task ID numbers:	0x1
Range of protected semaphore ID numbers:	0x1
Range of protected eventflag ID numbers:	0x1
Range of protected mailbox ID numbers:	0x1
Range of protected memory pool ID numbers:	0x1

2) System maximum value information

Task priority range:	0xf
Maximum number of tasks:	0x2
Maximum number of semaphores:	0x1
Maximum number of eventflags:	0x2
Maximum number of mailboxes:	0x3
Maximum number of memory pools:	0x2
Maximum number of cyclic handlers:	0x1
Maximum number of extended SVC handlers:	0x1
Maximum number of interrupt handlers:	0x5
Maximum interrupt source number:	0x30

3) System memory information

System Memory Pool 0:	Allocates a memory area for 0x2000 bytes, starting from .syspol0 section
System Memory Pool 1:	Allocates a memory area for 0x1000 bytes, starting from .syspol1 section
User Memory Pool 0:	Allocates a memory area for 0x7000 bytes, starting from .usrpol0_0 section
User Memory Pool 0:	Allocates a memory area for 0x2500 bytes, starting from .usrpol0_1 section
User Memory Pool 1:	Allocates a memory area for 0x1500 bytes, starting from .usrpol1 section

4) Task information

ID number:	0x1
Initial status:	ready
Activation code:	0x0
Extended information:	0x0
Description language:	Assembly language
Activation address:	_task01
Initial priority:	0x8
Interrupt mask status:	All interrupts enabled
Stack information for task:	Allocates a memory area for 0x100 bytes, starting from System Memory Pool 0
gp register-specific value:	Not set
tp register-specific value:	Not set
Key ID number:	0x1

ID number:	TASK02
Initial status:	dormant
Activation code:	0x0

Extended information:	0x0
Description language:	C language
Activation address:	_task02
Initial priority:	0xf
Interrupt mask status:	All interrupts disabled
Stack information for task:	Allocates a memory area for 0x100 bytes, starting from System Memory Pool 0
gp register-specific value:	Not set
tp register-specific value:	Not set
Key ID number:	0x2

5) Semaphore information

ID number:	0x1
Extended information:	0x0
Task queuing method:	Same order as that in which resource requests are issued (FIFO)
Initial resource count:	0xff
Maximum resource count:	0xff
Key ID number:	0x1

6) Eventflag information

ID number:	0x1
Extended information:	0x0
Whether waiting for multiple tasks:	Disable
Initial bit pattern:	0x0
Key ID number:	0x1

7) Mailbox information

ID number:	0x1
Extended information:	0x0
Task queuing method:	Same order as that in which message receive requests are issued (FIFO)
Message queuing method:	Same order as that in which messages are issued (FIFO)
Key ID number:	0x1

8) Indirectly activated interrupt handler information

Interrupt source number:	0x14 (INTP120)
Description language:	Assembly language
Activation address:	_inthdr01
gp register-specific value:	Not set
tp register-specific value:	Not set

9) Memory pool information

ID number:	0x1
Extended information:	0x0
Task queuing method:	According to task priority
Memory pool information:	Allocates a memory area for 0x2000 bytes, starting from User Memory Pool 0
Key ID number:	0x1

10) Cyclic handler information

Specification number:	0x1
Extended information:	0x0
Description language:	C language
Activation address:	_cychdr01
Initial activation status:	OFF status
Activation interval:	0x100 basic clock cycle
gp register-specific value:	Not set
tp register-specific value:	Not set

11) Extended SVC handler information

Extended function code:	0x1
Description language:	C language
Activation address:	_svchdr01
gp register-specific value:	Not set
tp register-specific value:	Not set

12) Initialization handler information

Description language:	C language
Activation address:	_varfunc

gp register-specific value:	Not set
tp register-specific value:	Not set

< SCT information >

1) Task management/task-associated synchronization management function system call information
Define the following as the task management/task-associated synchronization management function system call information used by a user processing program:

sta_tsk, exd_tsk

2) Synchronous communication (semaphore) management function system call information
Define the following as the synchronous communication (semaphore) management function system call information used by a user processing program:

sig_sem, wai_sem

3) Synchronous communication (eventflag) management function system call information
Define the following as the synchronous communication (eventflag) management function system call information used by a user processing program:

cre_flg, del_flg, set_flg, wai_flg

4) Synchronous communication (mailbox) management function system call information
Define the following as the synchronous communication (mailbox) management function system call information used by a user processing program:

cre_mbx, del_mbx, snd_msg, rcv_msg

5) Interrupt management function system call information
Define the following as the interrupt management function system call information used by a user processing program:

ena_int

6) Memory pool management function system call information
Define the following as the memory pool management function system call information used by a user processing program:

cre_mpl, del_mpl, get_blk, rel_blk

7) Time management function system call information
Define the following as the time management function system call information used by a user processing program:

act_cyc, ref_cyc

8) System management function system call information
Define the following as the system management function system call information used by a user processing program:

viss_svc

Figure 13-2 Example of System Configuration File

```
------------------------------
-- Declaration of the start of the Real-time OS information description
------------------------------
ser_def

------------------------------
-- Real-time OS information description
------------------------------

-- RX series information
rxsers      RX850PRO    V321

------------------------------
--Declaration of the start of the SIT information description
------------------------------
sit_def

------------------------------
-- SIT information description
------------------------------

-- System information
cputype         V850E1
clktim          0x1
clkhdr          INTCMD0
defstk          0x100
intstk          0x100:SPOL0
prtstk          0x1
prtsem          0x1
prtflg          0x1
prtmbx          0x1
prtmpl          0x1

-- System maximum value information
maxpri          0xf
maxtsk          0x2
maxsem          0x1
maxflg          0x2
maxmbx          0x3
maxmpl          0x2
maxcyc          0x1
maxsvc          0x1
maxint          0x5
maxintfactor    0x30

-- System memory information
mem         SPOL0       syspol0     0x2000
mem         SPOL1       syspol1     0x1000
mem         UPOL0       usrpol0_0   0x7000
mem         UPOL0       usrpol0_1   0x2500
mem         UPOL1       usrpol1     0x1500

-- Task information
tsk         0x1         TTS_RDY     0x0         0x0         TA_ASM      \
            _task01     0x8         TA_ENAINT   0x100:SPOL0 no_use      \
            no_use      0x1
tsk         TASK02      TTS_DMT     0x0         0x0         TA_HLNG     \
            _task02     0xf         TA_DISINT   0x100:SPOL0 no_use      \
            no_use      0x2

-- Semaphore information
sem         0x1         0x0         TA_TFIFO    0xff        0xff        0x1
```

```
-- Eventflag information
flg         0x1         0x0         TA_WSGL     0x0         0x1

-- Mailbox information
mbx         0x1         0x0         TA_TFIFO    TA_MFIFO    0x1

-- Indirectly activated interrupt handler information
inthdr      INTP120     TA_ASM      _inthdr01   no_use      no_use

-- Memory pool information
mpl         0x1         0x0         TA_TPRI     0x2000:UPOL0            0x1

-- Cyclic handler information
cyc         0x1         0x0         TA_HLNG     _cychdr01   TCY_OFF     \
            0x100       no_use      no_use

-- Extended SVC handler information
svc         0x1         TA_HLNG     _svchdr01   no_use      no_use

-- Initialization handler information
ini         TA_HLNG     _varfunc    no_use      no_use

------------------------------
-- Declaration of the start of the SCT information description
------------------------------
sct_def

------------------------------
-- SCT information description
------------------------------

-- Task management/task-associated synchronization management function system call
information
tsksvc      sta_tsk
tsksvc      exd_tsk

-- Synchronous communication (semaphore) management function system call information
semsvc      sig_sem
semsvc      wai_sem

-- Synchronous communication (eventflag) management function system call information
flgsvc      cre_flg
flgsvc      del_flg
flgsvc      set_flg
flgsvc      wai_flg

-- Synchronous communication (mailbox) management function system call information
mbxsvc      cre_mbx
mbxsvc      del_mbx
mbxsvc      snd_msg
mbxsvc      rcv_msg

-- Interrupt management function system call information
intsvc      ena_int

-- Memory pool management function system call information
mplsvc      cre_mpl
mplsvc      del_mpl
mplsvc      get_blk
mplsvc      rel_blk

-- Time management function system call information
timsvc      act_cyc
timsvc      ref_cyc
```

```
-- System management function system call information
syssvc      viss_svc
```

CHAPTER 14 CONFIGURATOR (CF850 Pro)

This chapter explains how to activate the configurator (CF850 Pro) and how information files (system information table file, system call table file, system information header file) are created.

14.1 Outline

To build systems (load modules) that use functions provided by the RX850 Pro, the information storing data to be provided for the RX850 Pro 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 RX850 Pro provides a utility tool (configurator CF850 Pro) that converts system configuration files which excel in descriptiveness and readability into information files.

The CF850 Pro reads system configuration files as input files, and then outputs information files.

The information files output from the CF850 Pro are explained below.

- System information table file
  An information file that stores data (resource information on the RX850 Pro such as the tasks, semaphores, and eventflags) required for the operation of the RX850 Pro.

- System call table file
  The system call table file stores data on types of system calls used in the processing program of the user.

- System information header file
  An information file that stores matching between ID numbers and object names (e.g. task, semaphore, and eventflag names) described in system configuration files

14.2 Activation Method

14.2.1 Activating from command line

The following is how to activate the CF850 Pro from the command line.
Note that, in the examples below, "C>" indicates the command prompt, and "Δ" indicates pressing of the space key.
The activation options enclosed in "[ ]" can be omitted.

C> cf850proΔ[@*cmd_file*]Δ[-cpuΔ*name*]Δ[-devpath=*path*]Δ[-iΔ*sit_file*]Δ[-cΔ*sct_file*]Δ[-dΔ*h_file*]Δ[-ni]Δ[-nc]Δ[-nd]Δ[-ne]Δ[-V]Δ[-help]Δ*cf_file*

The details of each activation option are explained below:

- *@cmd_file*
  Specifies the command file name.

  If omitted: The activation options specified on the command line is valid.

  Remark1 Specify the command file name "*cmd_file*" within 255 characters including the path name.

  Remark2 For the details on the command file, see "14.2.3 Command file".

- -cpuΔ*name*
  Specifies type specification names of target devices.

  If omitted: The CF850 Pro does not read device files. Therefore, in system configuration files, definitions using “interrupt factor names specified in device files” cannot be performed.

- -devpath=*path*
  Retrieves the device file corresponding to the target device specified with -cpuΔ*name* from the *path* folder.

  If omitted: The device file is retrieved in for the current folder.

- -iΔ*sit_file*
  Specifies the system information table file name to be output.

  If omitted: The CF850 Pro assumes that the following activation option is specified, and performs processing.

  -iΔsit.s

  Remark1 Specify the output file name "system information table file name: *sit_file*" within 255 characters including the path name.

  Remark2 When both this activation option and the -ni option are specified at the same time, only that which was input last is effective.

- -cΔ*sct_file*
  Specifies the system call table file name to be output.

  If omitted: The CF850 Pro assumes that the following activation option is specified, and performs processing.

  -cΔsct.s

  Remark1 Specify the output file name "system call table file name: *sct_file*" within 255 characters including the path name.

  Remark2 When both this activation option and the -nc option are specified at the same time, only that which was input last is effective.

- -dΔ*h_file*
  Specifies the system information header file name to be output.

  If omitted: The system changes the extension of the system configuration file name, specified with *cf_file*, to ".h", and outputs the file as the system information header file.

  Remark1 Specify the output file name "system information header file name: *h_file*" within 255 characters including the path name.

Remark2 When both this activation option and the -nd option are specified at the same time, only that which was input last is effective.

- -ni
  Disables output of the system information table file.

  If omitted: The CF850 Pro assumes that the following activation option is specified, and performs processing.

  -iΔsit.s

  Remark When both this activation option and the -iΔ*sit_file* option are specified at the same time, only that which was input last is effective.

- -nc
  Disables output of the system call table file.

  If omitted: The CF850 Pro assumes that the following activation option is specified, and performs processing.

  -cΔsct.s

  Remark When both this activation option and the -cΔ*sct_file* option are specified at the same time, only that which was input last is effective.

- -nd
  Disables output of the system information header file.

  If omitted: The system changes the extension of the system configuration file name, specified with *cf_file*, to ".h", and outputs the file as the system information header file.

  Remark When both this activation option and the -dΔ*h_file* option are specified at the same time, only that which was input last is effective.

- -ne
  Suppresses output of interrupt entries to the system information table file.

  If omitted: Interrupt entries are output to the system information table file.

- -V
  Outputs version information for the CF850 Pro to the standard output.

  If omitted: Version information for the CF850 Pro is not output.

  Remark Specifying this activation option nullifies all other activation options.

- -help
  Outputs the usage of the activation options for the CF850 Pro to the standard output.

  If omitted: The usage of the activation options for the CF850 Pro is not output.

  Remark Specifying this activation option nullifies all other activation options.

- *cf_file*
  Specifies the input file name "SYSTEM CONFIGURATION FILE name: *cf_file*" that input to the CF850 Pro.

  If omitted: This activation option cannot be omitted.

  Remark Specify the input file name "*cf_file*" within 255 characters including the path name.

14.2.2 Activating 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.

14.2.3 Command file

The CF850 Pro performs command file support from the objectives that eliminate specified probable activation option character count restrictions in the command lines.

Description formats of command files are described below.

1) Comment lines
Lines that start with # are treated as comment lines.

2) Activation options
For descriptions of different activation options, insert a space or line-feed code between them.
For descriptions of activation options composed of a parameter and a -*xxx* part such as -cpu, -i, -c, and -d, insert a space or line-feed code between the parameter and the -*xxx* part.

Remark When a parameter (whose path name is path) of -devpath has a folder name that contains a space code, the parameter must be enclosed in double quotes.

-devpath=”Program Files\DEV”

3) Maximum number of characters
The maximum number of characters that can be written to one single line in command files is 4096.

Figure 14-1 shows an example of command file description.
In the example of Figure 14-1 below, it is assumed that the following activation options are described.

Target device:	UPD70F3742
Reference folder for the device file:	C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile
System information table file:	sys.s (not including interrupt entries)
System call table file:	sit.s
System information header file:	sys.h
System configuration file:	sys.cfg

Figure 14-1 Example of Command File

```
# Command File
-cpu
f3742
-devpath="C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Device-
file"
-i
sis.s
-c
sct.s
-d
sys.h
-ne
sys.cfg
```

14.3 Command Input Examples

Examples of command input for the CF850 Pro are given below.
In this example, the UPD70F3742 is used as the target device.

- cf850pro -cpu f3742 -devpath=”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” -i sitfile.s -c sctfile.s -d hfile.h -ne cffile.cfg
  This command loads system configuration file cffile.cfg from the current folder, and the device file corresponding to device specification name f3742 from ”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” folder as input files, and then outputs the system information table file (not including interrupt entries) sitfile.s, system call table file sctfile.s, and system information header file hfile.h.

- cf850pro -cpu f3742 -devpath=”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” -i sitfile.s -ne cffile.cfg
  This command loads system configuration file cffile.cfg from the current folder, and the device file corresponding to device specification name f3742 from ”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” folder as input files, and then outputs the system information table file (not including interrupt entries) sit.s, system call table file sct.s, and system information header file cffile.h.

- cf850pro -cpu f3742 -devpath=”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” -c sctfile.s -ne cffile.cfg
  This command loads system configuration file cffile.cfg from the current folder, and the device file corresponding to device specification name f3742 from ”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” folder as input files, and then outputs the system information table file (not including interrupt entries) sit.s, system call table file sctfile.s, and system information header file cffile.h.

- cf850pro -cpu f3742 -devpath=”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” -d hfile.h -ne cffile.cfg
  This command loads system configuration file cffile.cfg from the current folder, and the device file corresponding to device specification name f3742 from ”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” folder as input files, and then outputs the system information table file (not including interrupt entries) sit.s, system call table file sct.s, and system information header file hfile.h.

- cf850pro -cpu f3742 -devpath=”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” -ne cffile.cfg
  This command loads system configuration file cffile.cfg from the current folder, and the device file corresponding to device specification name f3742 from ”C:\Program Files\NEC Electronics CubeSuite\CubeSuite\Device\V850\Devicefile” folder as input files, and then outputs the system information table file (not including interrupt entries) sit.s, system call table file sct.s, and system information header file cffile.h.

- cf850pro -V
  This command outputs version information for the CF850 Pro to the standard output.

- cf850pro -help
  This command outputs the usage of the activation options for the CF850 Pro to the standard output.

APPENDIX A　WINDOW REFERENCE

This appendix explains the window/panels that are used when the activation option for the CF850 Pro is specified from the integrated development environment platform CubeSuite.

A.1 Description

The following shows the list of window/panels.

Table A-1 List of Window/Panels

Window/Panel Name	Function Description
Main window	This is the first window to be open when CubeSuite is launched.
Project Tree panel	This panel is used to display the project components in tree view.
Property panel	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.

Main window

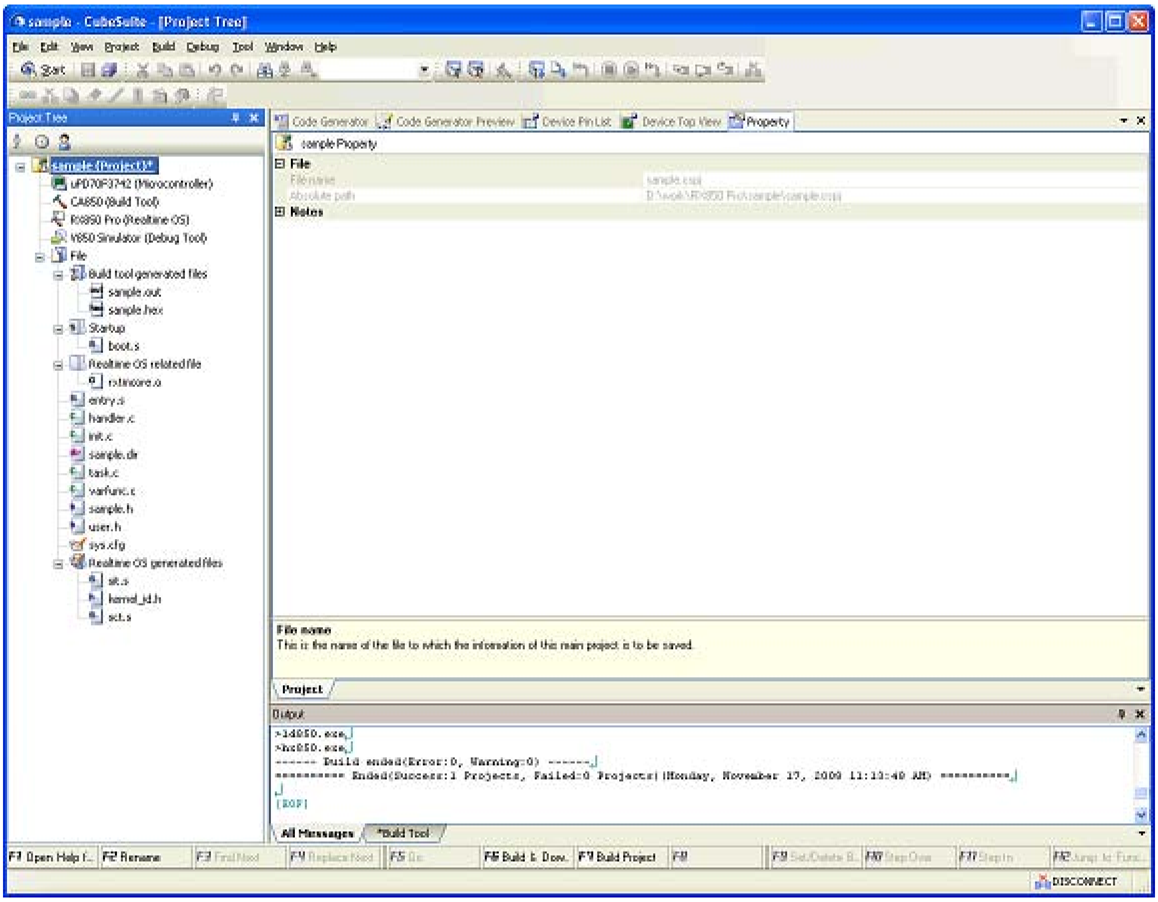
Outline

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] -> [NEC 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]

Realtime OS	The [View] menu shows the cascading menu to start the tools of realtime OS.
Resource Information	Opens the RD850Pro window. Note that this menu is disabled when the debug tool is not connected.
Performance Analyzer	Opens the AZ850 window. Note that this menu is disabled when the debug tool is not connected.

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

	Opens the RD850Pro window. 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.

Note　See CubeSuite V850 Build User's Manual for details about the Output panel.

Project Tree panel

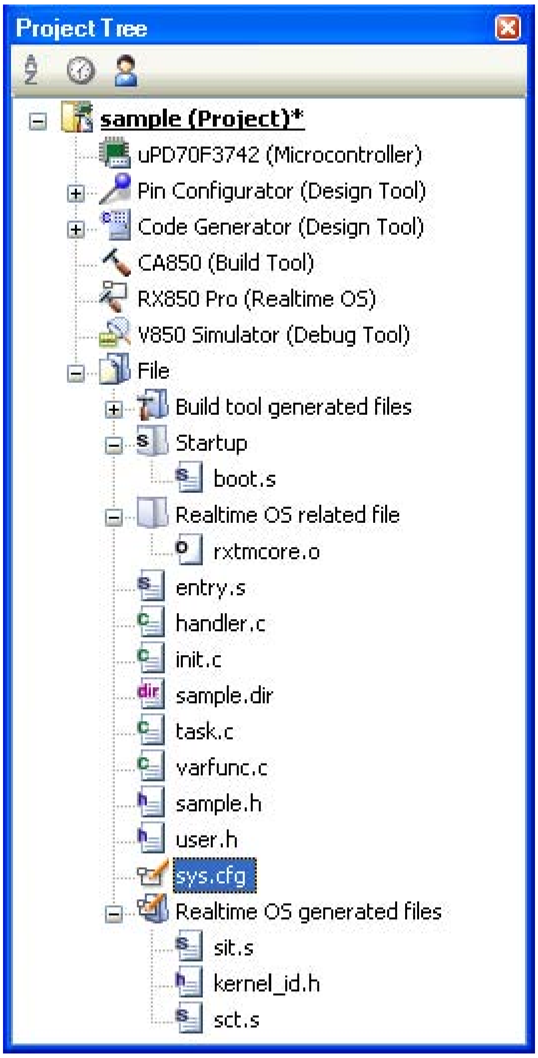
Outline

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



Explanation of each area

1) Project tree area

Project components are displayed in tree view with the following given node.

Node	Description
RX850 Pro(Realtime OS) (referred to as "Realtime OS node"))	Realtime OS to be used.
xxx.cfg	System configuration file.
Realtime OS related file (referred to as "Realtime OS related file node"))	The following object appear directly below the node created when project is added. - Nucleus common object (.o) This node and object displayed under this node cannot be deleted.
Realtime OS generated files (referred to as "Realtime OS generated files node"))	The following information files appear directly below the node created when a system configuration file is added. - System information table file (.s) - System information header file (.h) - System call table file (.s) 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, Realtime OS related file node, or Realtime OS generated files node is selected

Property	Displays the selected node's property on the Property panel.

2) When the nucleus common object, system configuration file, or an information file is selected

Assemble	Assembles the selected assembler source file. Note that this menu is only displayed when a system information table file or system call table file is selected. Note that this menu is disabled when the build tool is in operation.
Open	Opens the selected file with the application corresponds to the file extension. Note that this menu is disabled when multiple files are selected.
Open with Internal Editor...	Opens the selected file with the Editor panel. Note that this menu is disabled when multiple files are selected.
Open with Selected Application...	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.
Open Folder with Explorer	Opens the folder that contains the selected file with Explorer.
Add	Shows the cascading menu to add files and category nodes to the project.

Add File...	Opens the Add Existing File dialog box to add the selected file to the project.
Add New File...	Opens the Add File dialog box to create a file with the selected file type and add to the project.
Add New Category	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.
Remove from Project	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.
Copy	Copies the selected file to the clipboard. When the file name is in editing, the characters of the selection are copied to the clipboard.
Paste	This menu is always disabled.
Rename	You can rename the selected file. The actual file is also renamed.
Property	Displays the selected file's property on the Property panel.

Property panel

Outline

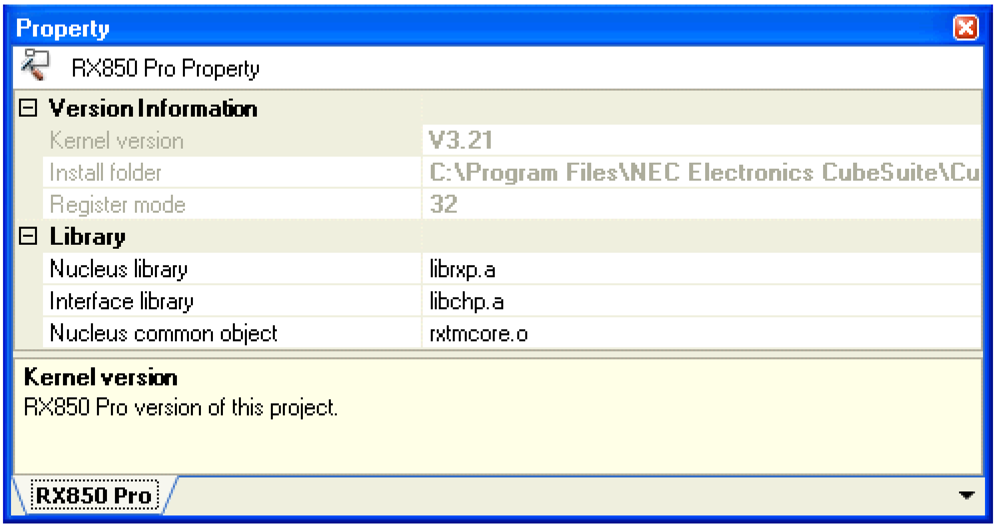
This panel is used to display the detailed information on the Realtime OS node, system configuration file, or the like that is selected on the Project Tree panel by every category and change the settings of the information.

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
  - [RX850 Pro] 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 related file node or Realtime OS generated files node is selected on the Project Tree panel
  - [Category Information] tab
- When the nucleus common object is selected on the Project Tree panel
  - [Build Settings] tab
  - [File Information] tab
- When the system information table file or system call table 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

Note 1 See CubeSuite V850 Build User's Manual for details about the [File Information] tab, [Category Information] tab, [Build Settings] tab, and [Individual Assemble Options] tab.

Note 2 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)

Undo	Cancels the previous edit operation of the value of the property.
Cut	While editing the value of the property, cuts the selected characters and copies them to the clip board.
Copy	Copies the selected characters of the property to the clip board.
Paste	While editing the value of the property, inserts the contents of the clip board.
Delete	While editing the value of the property, deletes the selected character string.
Select All	While editing the value of the property, selects all the characters of the selected property.

Context menu

Undo	Cancels the previous edit operation of the value of the property.
Cut	While editing the value of the property, cuts the selected characters and copies them to the clip board.

Copy	Copies the selected characters of the property to the clip board.
Paste	While editing the value of the property, inserts the contents of the clip board.
Delete	While editing the value of the property, deletes the selected character string.
Select All	While editing the value of the property, selects all the characters of the selected property.
Reset to Default	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.
Reset All to Default	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.

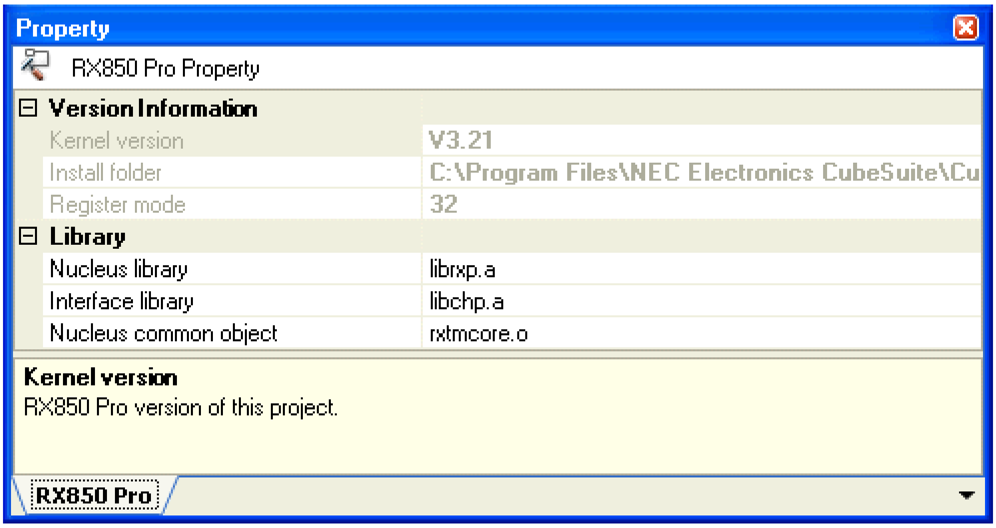
[RX850 Pro] tab

Outline

This tab shows the detailed information on RX850 Pro to be used categorized by the following.

- Version Information
- Library

Display image



Explanation of each area

1) [Version Information]

The detailed information on the version of the RX850 Pro are displayed.

Kernel version	Display the version of RX850 Pro to be used. Note that the version is set permanently when the project is created, and cannot be changed.	
	Default	*Using RX850 Pro version*
	How to change	Changes not allowed
Install folder	Display the folder in which RX850 Pro to be used is installed with the absolute path.	
	Default	*The folder in which RX850 Pro to be used is installed*
	How to change	Changes not allowed
Register mode	Display the register mode set in the project. Display the same value as the value of the [Select register mode] property of the build tool.	
	Default	*The register mode selected in the property of the build tool*
	How to change	Changes not allowed

Note An error occurs if the 26-register mode or 22-register mode in the [Select register mode] property of the build tool is selected. Although it is possible to run the build and create the load module with this selection, a warning is output.

2) [Library]

The detailed information on the library are displayed and the configuration can be changed.

<table>
<tr><td rowspan="5">Nucleus library</td><td colspan="3">Select the nucleus library which is referred when linking the application.</td></tr>
<tr><td>Default</td><td colspan="2">librxp.a</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="2">Restriction</td><td>librxp.a</td><td>Refers to librxp.a (a nucleus library that requires the first four bytes of the memory block to be zero-cleared when a rel_blk is issued).</td></tr>
<tr><td>librxpm.a</td><td>Refers to librxpm.a (a nucleus library that does not require the first four bytes of the memory block to be zero-cleared when a rel_blk is issued).</td></tr>
<tr><td rowspan="5">Interface library</td><td colspan="3">Select the interface library which is referred when linking the application.</td></tr>
<tr><td>Default</td><td colspan="2">libchp.a</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="2">Restriction</td><td>libchp.a</td><td>Refers to libchp.a (library with parameter check). All error codes are detected.</td></tr>
<tr><td>libncp.a</td><td>Refers to libncp.a (library without parameter check). Detects necessary error codes only.</td></tr>
<tr><td rowspan="5">Nucleus common object</td><td colspan="3">Select the nucleus common object which is linked to the user application.
If the object is changed, the name of the object displayed on the project tree.</td></tr>
<tr><td>Default</td><td colspan="2">rxtmcore.o</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="2">Restriction</td><td>rxtmcore.o</td><td>Links rxtmcore.o (the acceptance of interrupts that have higher priority than time interrupts is enabled in the cyclic handler).</td></tr>
<tr><td>rxcore.o</td><td>Links rxcore.o (the acceptance of all interrupts is enabled in the cyclic handler).</td></tr>
</table>

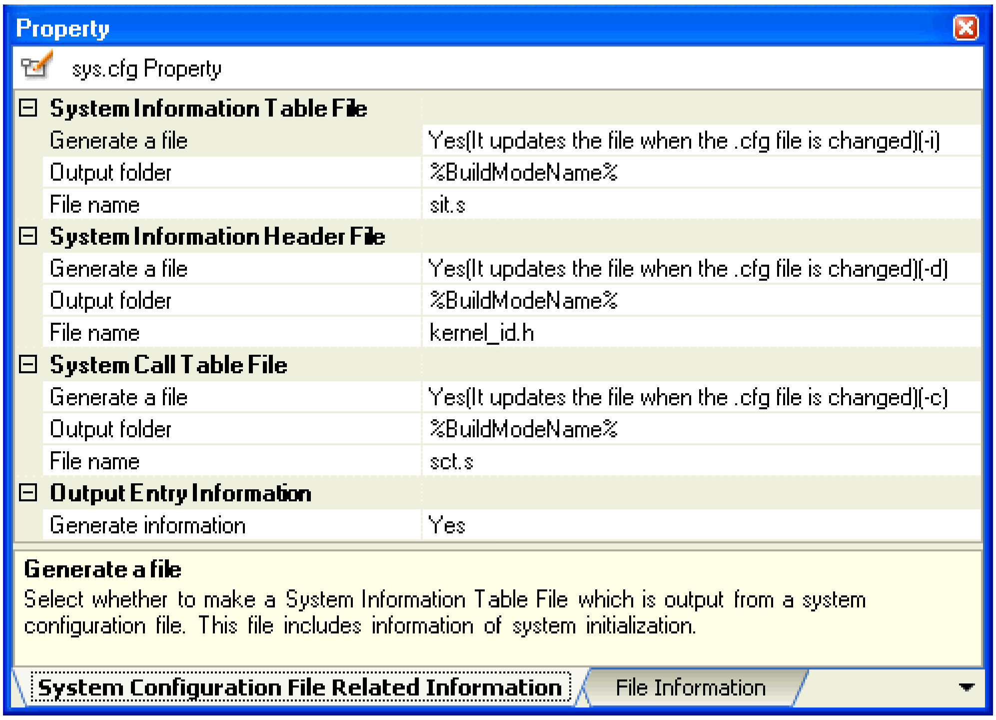
[System Configuration File Related Information] tab

Outline

This tab shows the detailed information on the using system configuration file categorized by the following and the configuration can be changed.

- System information table file
- System information header file
- System call table file
- Output entry information

Display image



Explanation of each area

1) [System Information Table File]

The detailed information on the system information table file are displayed and the configuration can be changed.

<table>
<tr><td rowspan="6">Generate a file</td><td colspan="3">Select whether to generate a system information table file and whether to update the file when the system configuration file is changed.</td></tr>
<tr><td>Default</td><td colspan="2">Yes(It updates the file when the .cfg file is changed)(-i)</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="3">Restriction</td><td>Yes(It updates the file when the .cfg file is changed)(-i)</td><td>Generates a new system information table file and displays it on the project tree.
If the system configuration file is changed when there is already a system information table file, then the system information table file is updated.</td></tr>
<tr><td>Yes(It does not update the file when the .cfg file is changed)(-ni)</td><td>Does not update the system information table file when the system configuration file is changed.
An error occurs during build if this item is selected when the system information table file does not exist.</td></tr>
<tr><td>No(It does not register the file to the project)(-ni)</td><td>Does not generate a system information table file and does not display it on the project tree.
If this item is selected when there is already a system information table file, then the file itself is not deleted.</td></tr>
<tr><td rowspan="4">Output folder</td><td colspan="3">Specify the folder for outputting the system information table file.
If a relative path is specified, the reference point of the path is the project folder.
If an absolute path is specified, the reference point of the path is the project folder (unless the drives are different).
The following macro name is available as an embedded macro.
%BuildModeName%: Replaces with the build mode name.
If this field is left blank, macro name "%BuildModeName%" will be displayed.
This property is not displayed when [No(It does not register the file that is added to the project)(-ni)] in the [Generate a file] property is selected.</td></tr>
<tr><td>Default</td><td colspan="2">%BuildModeName%</td></tr>
<tr><td>How to change</td><td colspan="2">Directly enter to the text box or edit by the Browse For Folder dialog box which appears when clicking the [...] button.</td></tr>
<tr><td>Restriction</td><td colspan="2">Up to 247 characters</td></tr>
<tr><td rowspan="4">File name</td><td colspan="3">Specify the system information table file name.
If the file name is changed, the name of the file displayed on the project tree.
Use the extension ".s". If the extension is different or omitted, ".s" is automatically added.
This property is not displayed when [No(It does not register the file that is added to the project)(-ni)] in the [Generate a file] property is selected.</td></tr>
<tr><td>Default</td><td colspan="2">sit.s</td></tr>
<tr><td>How to change</td><td colspan="2">Directly enter to the text box.</td></tr>
<tr><td>Restriction</td><td colspan="2">Up to 259 characters</td></tr>
</table>

2) [System Information Header File]

The detailed information on the system information header file are displayed and the configuration can be changed.

Generate a file	Select whether to generate a system information header file and whether to update the file when the system configuration file is changed.		
	Default	Yes(It updates the file when the .cfg file is changed)(-d)	
	How to change	Select from the drop-down list.	
	Restriction	Yes(It updates the file when the .cfg file is changed)(-d)	Generates a system information header file and displays it on the project tree. If the system configuration file is changed when there is already a system information header file, then the system information header file is updated.
		Yes(It does not update the file when the .cfg file is changed)(-nd)	Does not update the system information header file when the system configuration file is changed. An error occurs during build if this item is selected when the system information header file does not exist.
		No(It does not register the file to the project)(-nd)	Does not generate a system information header file and does not display it on the project tree. If this item is selected when there is already a system information header file, then the file itself is not deleted.
Output folder	Specify the folder for outputting the system information header file. If a relative path is specified, the reference point of the path is the project folder. If an absolute path is specified, the reference point of the path is the project folder (unless the drives are different). The following macro name is available as an embedded macro. %BuildModeName%: Replaces with the build mode name. If this field is left blank, macro name "%BuildModeName%" will be displayed. This property is not displayed when [No(It does not register the file that is added to the project)(-nd)] in the [Generate a file] property is selected.		
	Default	%BuildModeName%	
	How to change	Directly enter to the text box or edit by the Browse For Folder dialog box which appears when clicking the [...] button.	
	Restriction	Up to 247 characters	
File name	Specify the system information header file name. If the file name is changed, the name of the file displayed on the project tree. Use the extension ".h". If the extension is different or omitted, ".h" is automatically added. This property is not displayed when [No(It does not register the file that is added to the project)(-nd)] in the [Generate a file] property is selected.		
	Default	kernel_id.h	
	How to change	Directly enter to the text box.	
	Restriction	Up to 259 characters	

3) [System Call Table File]

The detailed information on system call table file are displayed and the configuration can be changed.

<table>
<tr><td rowspan="6">Generate a file</td><td colspan="3">Select whether to generate a system call table file and whether to update the file when the system configuration file is changed.</td></tr>
<tr><td>Default</td><td colspan="2">Yes(It updates the file when the .cfg file is changed)(-c)</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="3">Restriction</td><td>Yes(It updates the file when the .cfg file is changed)(-c)</td><td>Generates a system call table file and displays it on the project tree.
If the system configuration file is changed when there is already a system call table file, then the system information table file is updated.</td></tr>
<tr><td>Yes(It does not update the file when the .cfg file is changed)(-nc)</td><td>Does not update the system call table file when the system configuration file is changed.
An error occurs during build if this item is selected when the system call table file does not exist.</td></tr>
<tr><td>No(It does not register the file to the project)(-nc)</td><td>Does not generate a system call table file and does not display it on the project tree.
If this item is selected when there is already a system call table file, then the file itself is not deleted.</td></tr>
<tr><td rowspan="4">Output folder</td><td colspan="3">Specify the folder for outputting the system call table file.
If a relative path is specified, the reference point of the path is the project folder.
If an absolute path is specified, the reference point of the path is the project folder (unless the drives are different).
The following macro name is available as an embedded macro.
%BuildModeName%: Replaces with the build mode name.
If this field is left blank, macro name "%BuildModeName%" will be displayed.
This property is not displayed when [No(It does not register the file that is added to the project)(-nc)] in the [Generate a file] property is selected.</td></tr>
<tr><td>Default</td><td colspan="2">%BuildModeName%</td></tr>
<tr><td>How to change</td><td colspan="2">Directly enter to the text box or edit by the Browse For Folder dialog box which appears when clicking the [...] button.</td></tr>
<tr><td>Restriction</td><td colspan="2">Up to 247 characters</td></tr>
<tr><td rowspan="4">File name</td><td colspan="3">Specify the system call table file name.
If the file name is changed, the name of the file displayed on the project tree.
Use the extension ".s". If the extension is different or omitted, ".s" is automatically added.
This property is not displayed when [No(It does not register the file that is added to the project)(-nc)] in the [Generate a file] property is selected.</td></tr>
<tr><td>Default</td><td colspan="2">sct.s</td></tr>
<tr><td>How to change</td><td colspan="2">Directly enter to the text box.</td></tr>
<tr><td>Restriction</td><td colspan="2">Up to 259 characters</td></tr>
</table>

4) [Output Entry Information]

The detailed information on the output of entry information are displayed and the configuration can be changed.

<table>
<tr><td rowspan="5">Generate entry information</td><td colspan="3">Select whether to output the entry information into the system information table file.</td></tr>
<tr><td>Default</td><td colspan="2">Yes</td></tr>
<tr><td>How to change</td><td colspan="2">Select from the drop-down list.</td></tr>
<tr><td rowspan="2">Restriction</td><td>Yes</td><td>Outputs the entry information into the system information table file.
The file is not output when [No(It does not register the file that is added to the project)(-ni)] on the [Generate a file] property in the [System Information Table File] category is selected.</td></tr>
<tr><td>No(-ne)</td><td>Does not output the entry information into the system information table file.</td></tr>
</table>

APPENDIX B PROGRAMMING METHODS

This appendix explains how to describe processing programs when using the CA850 C compiler for the NEC Electronics V850 microcontrollers.

B.1 Outline

In the RX850 Pro, processing programs are classified according to purpose, as shown below.

- Task
  The minimum unit of a processing program that can be executed by the RX850 Pro.

- Directly activated interrupt handler
  A routine dedicated to interrupt processing. When an interrupt occurs, this handler is activated without using the RX850 Pro.
  Because the RX850 Pro does not intervene, it cannot issue system calls in the handler, but the response speed is expected to be high.

- Indirectly activated interrupt handler
  A routine dedicated to interrupt processing. When an interrupt occurs, this handler is activated upon the completion of the interrupt preprocessing by the RX850 Pro (such as saving the contents of the registers or switching the stack).
  Because interrupt preprocessing is performed by the RX850 Pro, the indirectly activated interrupt handler has an advantage in that system calls can be issued in the handler, despite response speed being degraded compared with the directly activated interrupt handler.

- Cyclic handler
  A routine dedicated to cyclic processing. Every time the specified time elapses, this handler is activated immediately. This routine is handled independently of tasks. When the activation time has been reached, therefore, the processing of the task currently being executed is canceled even if that task has the highest priority relative to all other tasks in the system, and control is passed to the cyclic handler.
  A cyclic handler incurs a smaller overhead before the start of execution, relative to any other cyclic processing programs written by the user.

- Extended SVC handler
  A function registered by the user as an extended system call.
  These processing programs have their own basic formats according to the general conventions or conventions to be applied when the RX850 Pro is used.

B.2 Keywords

The character strings listed below are reserved as keywords for the configurator. These strings cannot, therefore, be used for other purposes.

clkhdr, clktim, cyc, defstk, flg, flgsvc, ini, inthdr, intstk, intsvc, maxcyc, maxflg, maxint, maxintfactor, maxmbx, maxmpl, maxpri, maxsem, maxsvc, maxtsk, mbx, mbxsvc, mem, mpl, mplsvc, no_use, prtflg, prtmbx, prtmpl, prtsem, prttsk, RX850PRO, rxsers, sct_def, sem, semsvc, ser_def, sit_def, SPOL0, SPOL1, svc, syssvc, TA_ASM, TA_DISINT, TA_ENAINT, TA_HLNG, TA_MFIFO, TA_MPRI, TA_TFIFO, TA_TPRI, TA_WMUL, TA_WSGL, TCY_OFF, TCY_ON, timsvc, tsk, tsksvc, TTS_DMT, TTS_RDY, UPOL0, UPOL1

B.3 Reserved Words

The character strings listed below are reserved as external symbols for the RX850 Pro. These strings cannot, therefore, be used for other purposes.

`_x_, _f_, _e_, _rx_`

Remark The use of these character strings is prohibited when a single load module is created. There is no problem if a symbol starting with any of these character strings is used when a load module that separates the RX850 Pro and application is created.

B.4 Hardware Status When Processing Program Is Activated

The hardware statuses (indicated by the ID bit of sp, tp, gp, ep, and psw) when the processing program is activated are listed below.

Table B-1 Hardware Status (Task)

	Task
Stack pointer (sp)	Task stack (value in the pool area specified during task generation)
Text pointer (tp)	Value given when __rx_start is called
Global pointer (gp)	alue specified during task generation (undefined if no_use is specified)
Element pointer (ep)	alue specified during task generation (undefined if no_use is specified)
Interrupt status (ID bit of psw)	Value specified during task generation (default: Interrupts are enabled)

Table B-2 Hardware Status (Directly Activated Interrupt Handler)

	Directly Activated Interrupt Handler
Stack pointer (sp)	Stack when an interrupt occurs
Text pointer (tp)	Undefined
Global pointer (gp)	Value given when an interrupt occurs
Element pointer (ep)	Value given when an interrupt occurs
Interrupt status (ID bit of psw)	Interrupts are disabled

Table B-3 Hardware Status (Indirectly Activated Interrupt Handler)

	Indirectly Activated Interrupt Handler
Stack pointer (sp)	System stack (value in the pool area specified during system stack definition)
Text pointer (tp)	Undefined (value given when __rx_start is called)
Global pointer (gp)	Value specified during generation of indirectly activated interrupt handler (undefined if no_use is specified)
Element pointer (ep)	Value specified during generation of indirectly activated interrupt handler (undefined if no_use is specified)
Interrupt status (ID bit of psw)	Interrupts are disabled

Table B-4 Hardware Status (Cyclic Handler)

	Cyclic Handler
Stack pointer (sp)	System stack (value in the pool area specified during system stack definition)
Text pointer (tp)	Undefined (value given when __rx_start is called)
Global pointer (gp)	Value specified during cyclic handler generation (undefined if no_use is specified)
Element pointer (ep)	Value specified during cyclic handler generation (undefined if no_use is specified)
Interrupt status (ID bit of psw)	Interrupts are disabled

Table B-5 Hardware Status (Extended SVC Handler)

	Extended SVC Handler
Stack pointer (sp)	Stack when an extended SVC handler is called
Text pointer (tp)	Undefined (value given when __rx_start is called)
Global pointer (gp)	Value specified during extended SVC handler generation (undefined if no_use is specified)
Element pointer (ep)	Value specified during extended SVC handler generation (undefined if no_use is specified)
Interrupt status (ID bit of psw)	Status when an extended SVC handler is called

Table B-6 Hardware Status (Initialization Handler)

	Initialization Handler
Stack pointer (sp)	System stack (value in the pool area specified during system stack definition)
Text pointer (tp)	Undefined (value given when __rx_start is called)
Global pointer (gp)	Value specified during initialization handler generation (undefined if no_use is specified)
Element pointer (ep)	Value specified during initialization handler generation (undefined if no_use is specified)
Interrupt status (ID bit of psw)	Interrupts are disabled

B.5 Tasks

When describing a task in C language, describe it as a void-type function having one INT-type argument after function declaration by pragma directive.

An activation code that is specified in Task information during configuration or an activation code that is specified upon issuance of sta_tsk is specified for the argument (stacd).

Figure B-1 shows the task description format (in C language).

Figure B-1 Task (C Language)

```
#include    <stdrx85p.h>

#pragma     rtos_task   func_task

void
func_task ( INT stacd )
{
            /*Processing of task func_task*/
            ...............
            ...............

            /*Termination of task func_task*/
            ext_tsk ( );
}
```

Remark See CubeSuite Coding User's Manual for details about the function declaration by pragma directive.

When describing a task in assembly language, describe it as a function conforming to the function call conventions of the CA850.

An activation code that is specified in Task information during configuration or an activation code that is specified upon issuance of sta_tsk is specified for the argument (r6 register).

Figure B-2 shows the task description format (in assembly language).

Figure B-2 Task (Assembly Language)

```
.include    "stdrx85p.inc"

            .text
            .align      4
            .globl      _func_task
_func_task :
            #Processing of task func_task
            ...............
            ...............

            #Termination of task func_task
            jr          _ext_tsk
```

B.6 Directly Activated Interrupt Handler

B.6.1 Recommended

Use C (using pragma or the like) or assembly language for coding directly activated interrupt handlers.
See the hardware user's manual for the V850 microcontroller used or CubeSuite Coding User's Manual for details.

B.6.2 To implement functionsequivalent to indirectly activated interrupt handler

Using the directly activated interrupt handler, the functions equivalent to the indirectly activated interrupt handler can be implemented (such as enabling calling of system calls, etc.).
Assembly language is used for implementation, but it is also possible to code the main processing in C and calling it using the Jarl instruction.
The register data must be saved before processing for implementation, and restored after the processing.
However, the RX850 Pro provides a macro that performs saving and restoring the register data, which reduces the load on the user in writing the handlers in assembly language.
If the functions equivalent to the indirectly activated interrupt handler are implemented using the directly activated interrupt handler, the functions are equivalent but the response speed is degraded. In addition, the definition method is more complicated than that for the indirectly activated interrupt handler.
That is, there are a few merits in terms of function and performance but there is a demerit in terms of definition complexity, compared with the indirectly activated interrupt handler.
To use the RX850 Pro functions such as interrupt handler system calls, usually the use of indirectly activated interrupt handler is therefore recommended.
The following figure shows the description format (in assembly language) of the directly activated interrupt handler when the CA850 is used.

Figure B-3 Directly Activated Interrupt Handler (Assembly Language)

```
.include    "stdrx85p.inc"

            /*Interrupt entry*/
            .section    "int_name", text
            jr          _func_inthdr

            .text
            .align      4
            .globl      _func_inthdr
_func_inthdr :
            /*Saving registers, switching stack*/
            RTOS_IntEntry

            /*Main processing of directly activated interrupt handler*/
            .extern     _inthdr_body
            jarl        _inthdr_body, lp

            /*r10: ID of task to be woken up after returning from handler*/
            /*Switching stack, restoring registers*/
            /*Return from directly activated interrupt handler and waking up task*/
            RTOS_IntReturnWakeup    r10
```

```
#include    <stdrx85p.h>

ID
inthdr_body ( void )
{
            __asm ( "mov #__tp_TEXT, tp" );
            __asm ( "mov #__gp_DATA, gp" );

            /*Processing of directly activated interrupt handler func_inthdr*/
            ...............
            ...............

            /*Return from directly activated interrupt handler func_inthdr*/
            return ( tskid );
}
```

First, describe the interrupt handler entry processing (jr instruction) at the handler address. Refer to the second and third rows in this example.

Next, describe the interrupt handler main unit processing.

The macro RTOS_IntEntry notifies the RX850 Pro of the activation of the handler, the saving of the temporary register and lp, and the switching of the task. The other registers (r20 to r30) are then saved, and control is transferred to the handler. In the above example, the C function, inthdr_body, of the handler is called. Before the execution of the handler main unit processing, set the tp (text pointer) and gp (global pointer) used by the handler.

As described in "6.3 Directly Activated Interrupt Handler", the values of the gp and tp become undefined. Since this setting must be described in assembly language, use the __asm instruction as in the above example or the #pragma asm to pragma endasm directives to describe the handler in C language. In the handler, "the system calls that can be issued from the handler" explained in the user's manual can be issued.

When the issuance processing of the handler is completed, the registers saved by the user must be restored and execution must return from the interrupt handler. To wake up a task specified after execution has returned from an interrupt, the ID of the task to be woken up must be set to register r10. In the above example, a task ID is returned as a return value when execution returns from inthdr_body, and its value is copied to r10. This operation is performed with the code output from the CA850.

The reti instruction can also be used to return from the interrupt through simple processing. At that time, data in the register must be restored before issuing the instruction.

Remark Set a branch instruction that branches to the directly activated interrupt handler at the handler address to which the processor transfers control if an interrupt occurs. This is done by the .section quasi directive in Figure B-3.
See CubeSuite Coding User's Manual for details about the .section quasi directive. Specify an interrupt request name defined in the device file as "int_name".

B.7 Indirectly Activated Interrupt Handler

When describing an indirectly activated interrupt handler in C language, describe it as an ID-type function having no argument.

Figure B-4 shows the description format of an indirectly activated interrupt handler (in C language).

Figure B-4 Indirectly Activated Interrupt Handler (C Language)

```
#include    <stdrx85p.h>

ID
func_inthdr ( void )
{
        /*Processing of indirectly activated interrupt handler func_inthdr*/
            ...............
            ...............

       /*Return processing from indirectly activated interrupt handler func_inthdr*/
            return ( TSK_NULL );
}
```

Remark An indirectly activated interrupt handler is a subroutine called by interrupt processing in the nucleus. Therefore, when an indirectly activated interrupt handler is described, an instruction for branching to the indirectly activated interrupt handler needs to be set for the handler address to which the processor passes control upon the occurrence of an interrupt. This setting must be described in assembly language.
However, because the RX850 Pro provides the processing that should be described as the branch instruction in the form of a macro, this macro should be used. For example, to use the INTP100 (address: 0x100) maskable interrupt as an indirectly activated interrupt handler, describe as follows.

```
.section    "INTP100"
RTOS_IntEntry_Indirect
```

The same description is required for clock interrupts since they are handled as indirectly activated interrupt handlers.

When describing an indirectly activated interrupt handler in assembly language, describe it as a function conforming to the function call conventions of the CA850.

Figure B-5 shows the description format of an indirectly activated interrupt handler (in assembly language) when the CA850 is used.

Figure B-5 Indirectly Activated Interrupt Handler (Assembly Language)

```
.include    "stdrx85p.inc"

            .text
            .align      4
            .globl      _func_inthdr

_func_inthdr :
        #Processing of indirectly activated interrupt handler func_inthdr
            ...............
            ...............

        #Return processing from indirectly activated interrupt handler func_inthdr
            mov         TSK_NULL, r10
            jmp         [lp]
```

Remark An indirectly activated interrupt handler is a subroutine called by interrupt processing in the nucleus. Therefore, when an indirectly activated interrupt handler is described, an instruction for branching to the indirectly activated interrupt handler needs to be set for the handler address to which the processor passes control upon the occurrence of an interrupt. This setting must be described in assembly language.
However, because the RX850 Pro provides the processing that should be described as the branch instruction in the form of a macro, this macro should be used. For example, to use the INTP100 (address: 0x100) maskable interrupt as an indirectly activated interrupt handler, describe as follows.

```
.section    "INTP100"
RTOS_IntEntry_Indirect
```

The same description is required for clock interrupts since they are handled as indirectly activated interrupt handlers.

B.8 Cyclic Handler

When describing a cyclic handler in C language, describe it as a void-type function having no argument.
Figure B-6 shows the description format of a cyclic handler (in C language).

Figure B-6 Cyclic Handler (C Language)

```
#include     <stdrx85p.h>

void
func_cychdr ( void )
{
            /*Processing of cyclic handler func_cychdr*/
            ...............
            ...............

            /*Return processing from cyclic handler func_cychdr*/
            return;
}
```

Remark A cyclic handler is a subroutine called by system clock processing in the nucleus.

When describing a cyclic handler in assembly language, describe it as a function conforming to the function call conventions of the CA850.
Figure B-7 shows the description format of a cyclic handler (in assembly language) when the CA850 is used.

Figure B-7 Cyclic Handler (Assembly Language)

```
.include     "stdrx85p.inc"

            .text
            .align      4
            .globl      _func_cychdr
_func_cychdr :
            #Processing of cyclic handler func_cychdr
            ...............
            ...............

            #Return processing from cyclic handler func_cychdr
            jmp         [lp]
```

Remark A cyclic handler is a subroutine called by system clock processing in the nucleus.

B.9 Extended SVC Handler

When describing an extended SVC handler in C language, describe it as an INT-type function.
Figure B-8 shows the description format of an extended SVC handler (in C language).

Figure B-8 Extended SVC Handler (C Language)

```
#include    <stdrx85p.h>

INT
func_svchdr ( VW prm1, VW prm2, VW prm3 )
{
            int         ret;

            /*Processing of extended SVC handler func_svchdr*/
            ...............
            ...............

            /*Return processing from extended SVC handler func_svchdr*/
            return ( INT ret );
}
```

When describing an extended SVC handler in assembly language, describe it as a function conforming to the function call conventions of the CA850.
Figure B-9 shows the description format of an extended SVC handler (in assembly language).

Figure B-9 Extended SVC Handler (Assembly Language)

```
.include    "stdrx85p.inc"

            .text
            .align      4
            .globl      _func_svchdr
_func_svchdr :
            #Processing of extended SVC handler func_svchdr
            ...............
            ...............

            #Return processing from extended SVC handler func_svchdr
            mov         ret, r10
            jmp         [lp]
```

APPENDIX C MEMORY AND MEMORY CAPACITY ESTIMATION

This chapter explains how the RX850 Pro manages the memory (RAM), and the memory capacity used.

C.1 SPOL and UPOL

The RX850 Pro uses the following 4 RAM areas:

SPOL0: System Memory Pool 0
SPOL1: System Memory Pool 1
UPOL0: User Memory Pool 0
UPOL1: User Memory Pool 1

The location information of these memory areas is determined by specifying their "first address" and "size" in the system configuration file. In other words, the addresses and size of the usable RAM areas must be specified.
The usage of these memory pools are predetermined as indicated in the table below.

Table C-1 Types of Memory Pools and Assigned Items

Memory Pool Name	Assigned Items
SPOL0	Operating system management table (SBT) Ready queue Management blocks Stack for task Stack for interrupt handler
SPOL1	Stack for task Stack for interrupt handler Memory pool
UPOL0	Memory pool
UPOL1	Memory pool

SPOL0 must always be generated because information on the system of the RX850 Pro is located in this memory pool. SPOL1 does not have to be generated if SPOL0 suffices. It is possible to improve the performance of the system by locating SPOL0, in which management blocks are located, in the internal RAM, and SPOL1, which requires a relatively large size, in the external RAM.
UPOL1 is necessary for using the memory management function of the RX850 Pro. In this case, also create UPOL0. It is not possible to create just UPOL1.

C.2 Memory Capacity in Management Area

This section explains the size used by the operating system management table and management blocks of the RX850 Pro. The operating system management table and management blocks are reserved from SPOL0. Table C-2 shows the size of a management area used per object and how to calculate the size.

Table C-2 Size of Object Management Area

Object Name	Management Area Size Per Object (in bytes)	Size Calculation Method (in bytes)
Operating system management table, ready queue	520 to 1048	504 + align 32 (Task priority range + 4) / 8 + align 4 ((Task priority range + 4) * 2)
		“Task priority range" is the value of *pri_lvl* of the System maximum value information specified during configuration.
System memory area management block	8	8 * 4 = 32
		8 bytes for SPOL0, SPOL1, UPOL0, and UPOL1 each. Even when all the 4 memory pools are not created, 32 bytes are always reserved because 4 tables are always reserved.
Task management block	56	56 * Maximum number of tasks
		"Maximum number of tasks" is the value of *tsk_cnt* of the System maximum value information specified during configuration.
Semaphore management block	20	20 * Maximum number of semaphores
		"Maximum number of semaphores" is the value of *sem_cnt* of the System maximum value information specified during configuration.
Eventflag management block	20	20 * Maximum number of eventflags
		"Maximum number of eventflags" is the value of *flg_cnt* of the System maximum value information specified during configuration.
Mailbox management block	20	20 * Maximum number of mailboxes
		"Maximum number of mailboxes" is the value of *mbx_cnt* of the System maximum value information specified during configuration.
Interrupt handler management block	16	16 * Maximum number of interrupt handlers + align 4 (Maximum interrupt source number)
		"Maximum number of interrupt handlers" is the value of *ith_cnt* of the System maximum value information specified during configuration. "Maximum interrupt source number" is the value of *itf_cnt* of the System maximum value information specified during configuration.
Cyclic handler management block	40	40 * Maximum number of cyclic handlers
		"Maximum number of cyclic handlers" is the value of *cyc_cnt* of the System maximum value information specified during configuration.
Memory pool management block	24	24 * Maximum number of memory pools
		"Maximum number of memory pools" is the value of *mpl_cnt* of the System maximum value information specified during configuration.

Object Name	Management Area Size Per Object (in bytes)	Size Calculation Method (in bytes)
Extended SVC handler management block	16	16 * Maximum number of extended SVC handlers
		"Maximum number of extended SVC handlers" is the value of *svc_cnt* of the System maximum value information specified during configuration.

C.3 Capacity of Task Stack

The task stack area is classified into the following 4 areas:

- Stack management block
- Context area
- Interrupt stack frame
- Task area

When generating a task (during configuration or when cre_tsk is issued), the task stack size must be specified. The specified value is the total size of the interrupt stack frame and the area used for tasks. For the stack size actually secured on the memory, the sizes of the stack management table and context area and the value aligned to 4 bytes are also added.

The size of the "task area" varies depending on the user application. However, the size of the "stack management block", "context area", and "interrupt stack frame" is predetermined, as follows.

Table C-3 Size Used for Task Stack

Task Stack Area	Size (in bytes)
Stack management block	28
Context area (Interrupt stack frame: 72 bytes)	148
Task area	Depends on application

When a task is generated (during configuration or when cre_tsk is issued) and 100 bytes is specified for the task stack size, the stack size actually secured on the memory is as follows.

100 + 28 + 148 = 276 bytes

If the extended SVC handler is started from a task, an area in which registers are saved for handler execution and the stack area consumed by the SVC handler are necessary. The size of these areas is as follows.

Table C-4 Size of Task Stack Used for Extended SVC Handler

Task Stack Area	Size (in bytes)
Register saving area for extended SVC handler	28
Extended SVC handler area	Depends on application

The task stack area is reserved from SPOL0 or SPOL1 when a task is created, and is released when the task is deleted (by del_tsk, exd_tsk, or ter_tsk). If there is a possibility that all tasks could be simultaneously created, therefore, the size of each task must be calculated and the size of SPOL0 and SPOL1 must be determined so that the total size of all the tasks can be reserved. If all tasks are not created at the same time, calculate the maximum size of the combination of tasks that are created at the sometime, and determine the size of SPOL0 and SPOL1 based on this size.

Next, the method for calculating the task stack size is summarized. The total size of the all items is the size that must be secured as the memory area, and the total size of the shaded portions shall be specified as the task stack size when a task is generated (during configuration or when cre_tsk is issued). The value secured for the memory size must be aligned to 4 bytes.

Table C-5 Summary of Size Used for Task Stack

Task Stack Area	Size (in bytes)	Remark
Stack management block	28	-
Context area (Interrupt stack frame: 72 bytes)	148	-
Task area	Depends on application	Calculate and specify size of stack where tasks are pushed and popped. Take the number of variables used into consideration. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from a task.
Register saving area extended SVC handler	28	Unnecessary if the extended SVC handler is not used
Extended SVC handler area	Depends on application	Unnecessary if the extended SVC handler is not used. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from the extended SVC handler.

C.4 Capacity of Stack for Interrupt Handler

The stack area for the interrupt handler is used by the following 4 handlers and task.

- Initialization handler
- Idle task
- Interrupt handler
- Cyclic handler

The size used by each handler or task is explained below.

1) Initialization handler
Reserve an area of the size consumed by functions (for pushing and popping) described as the initialization handler. While the initialization handler is being executed, no task or interrupt handler is started. If the consumed size is less than the area size explained in 2 below, therefore, the stack size consumed by the initialization handler does not have to taken into account.

2) Idle task
A stack area is consumed by an interrupt that occurs before the next task is executed while an idle task is being executed or after a task has been terminated (by issuance of del_tsk, exd_tsk, or ter_tsk). The size of this area is 72 bytes. In some cases, the more stack area may be necessary. In those cases, refer to “3) Interrupt handler” below and the sections that follow.
Because this 72-byte stack area is added during initialization processing, it does not have to be taken into account during configuration. This means that the memory size actually reserved is the stack area for interrupt handlers specified during configuration plus 72 bytes.

3) Interrupt handler
When an interrupt is generated for the first time (when a task is interrupted), an interrupt handler is generated in a task stack or an area for idle tasks described in "2) Idle task". If multiple interrupts may occur after that, the interrupt stack frame size multiplied by the maximum nest count must be added.
To activate an interrupt handler, an additional 28 bytes must be secured as the register data save area, separately from the interrupt stack frame size. This is the total amount that, for example, after information of the interrupt stack frame is stored in a task stack, the stack pointer (sp) points to the stack for the interrupt handler, and additional data is stored. This size must be considered in a system in which multiple interrupts are enabled. Therefore, multiply 28 by "the maximum interrupt nest count + 1 (for the first interrupt)" and add the value to the stack size used by the interrupt handler.
Moreover, add the additional stack size by making allowances for the case where a function that is used as an interrupt handler consumes the stack for pushing or popping the stack elements, and interrupt nests are at the maximum count. That is, add 4 bytes when issuing system calls in the interrupt handler, or 28 bytes when issuing an extended SVC handler. Add an additional 4 bytes if system calls are issued in the extended SVC handler.
The clock handler is treated as an interrupt handler that does not consume the stack by pushing and popping. Calculate the stack size taking this into consideration.

4) Cyclic handler
The cyclic handler is provided as a subroutine that is called by the clock handler.
If another cyclic handler is started because a new clock interrupt occurs while 1 cyclic handler is being executed, the processing of the cyclic handler already under execution takes precedence. Therefore, add the maximum stack size consumed by the function of all the functions described as a cyclic handler to the size of the handler stack.

The table below summarizes the methods of calculating the size of the stack for interrupt handlers. The total size must be reserved as a memory area, and the total of the shaded sizes is the size for the interrupt handler specified during configuration. Note that the value reserved on the memory area is aligned to 4 bytes.

Table C-6 Stack Size for Interrupt Handler in System Not Enabling Multiple Interrupts

Interrupt Handler Stack Area	Size (in bytes)	Remarks
Idle task area	144	-
Register saving area for interrupt handler	28	-
Interupt handler area	Depends on application	Calculate and specify the size of the stack where the interrupt handler pushes and pops. Take the number of variables used into consideration. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from an interrupt handler. Specify the stack size used by the handler (including a cyclic handler) that uses the stack most of all the interrupt handlers used.
Register saving area for extended SVC handler\	28	Unnecessary if an interrupt handler does not call the extended SVC handler.
Extended SVC handler area	Depends on application	Unnecessary if an interrupt handler does not call the extended SVC handler. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from the extended SVC handler.

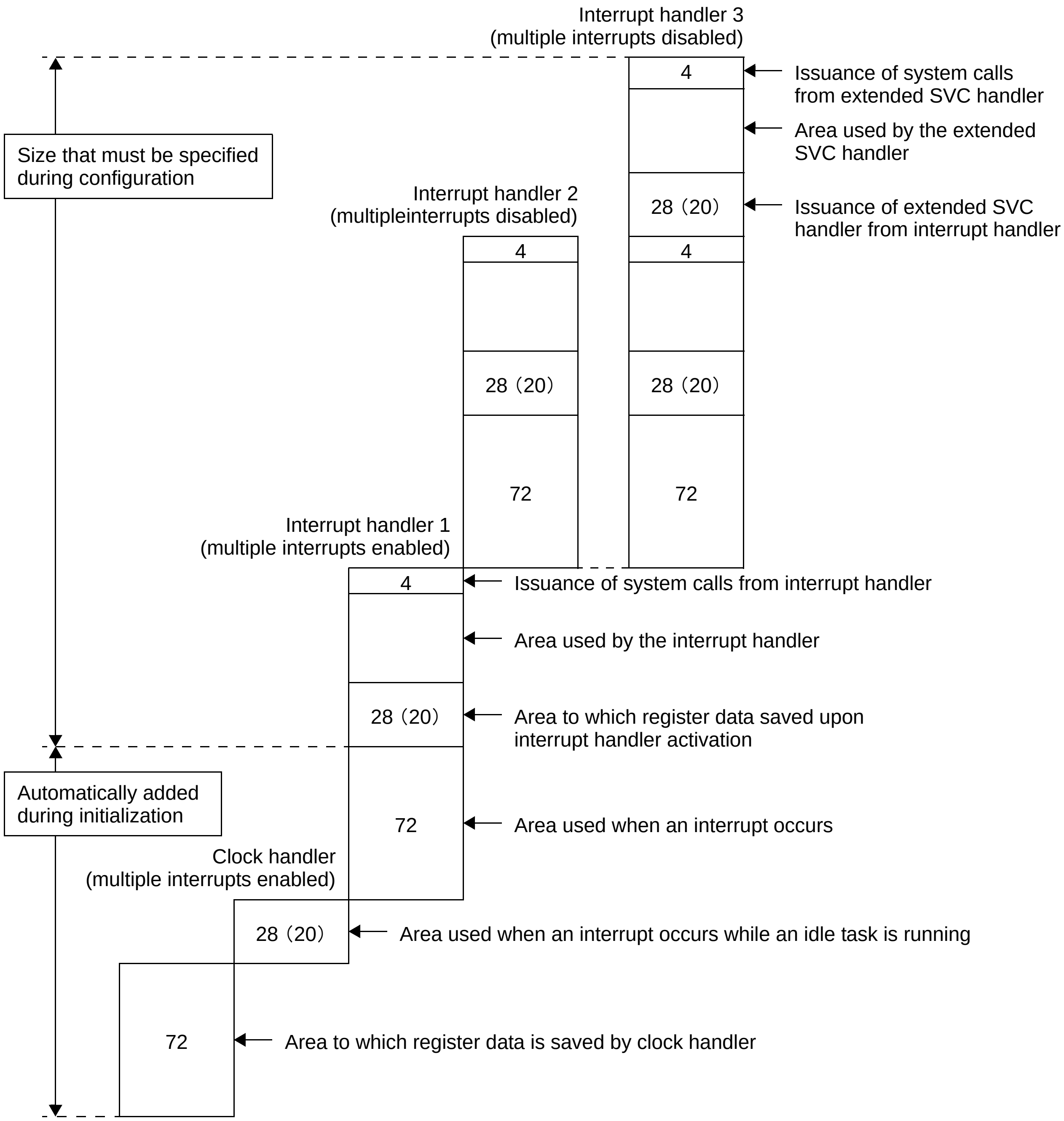
This table indicates the stack size used when multiple interrupts are not enabled. If the cyclic handler is interrupted and if that interrupt is acknowledged, this is equivalent to multiple interrupts. In other words, the stack size in this table applies to an application that is executed when rxtmcore.o (version that can acknowledge an interrupt with a higher priority than the clock interrupt in the cyclic handler) is used as the nucleus common object, when an interrupt with a priority higher than that of the clock interrupt is not used, and when all the interrupt handlers are disabled.

Table C-7 Stack Size for Interrupt Handler in System Enabling Multiple Interrupts

Interrupt Handler Stack Area	Size (in bytes)	Remarks
Idle task area	144	-
Interrupt stack frame	72 * n	-
Register saving area for interrupt handler	28 * (n+1)	-
Interupt handler area	Depends on application	Calculate and specify the size of the stack where the interrupt handler pushes and pops. Take the number of variables used into consideration. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from an interrupt handler. Specify the stack size used by the handler (including a cyclic handler) that uses the stack most of all the interrupt handlers used.
Register saving area for extended SVC handler	28 * m	Unnecessary if an interrupt handler does not call the extended SVC handler.
Extended SVC handler area	Depends on application	Unnecessary if an interrupt handler does not call the extended SVC handler. Add 4 bytes because the RX850 Pro pushes lp (r31) when a system call is issued from the extended SVC handler.

In the above table, n indicates the maximum number of times interrupts are nested, and m indicates the number of interrupt handlers using the extended SVC handler.

Figure C-1 Estimation of Stack Area for Interrupt Handlers



C.5 Memory Pool Capacity

The following describes the capacity of memory areas (system memory, memory pool, and memory block).
The memory area is secured using the procedure as follows:

- Secure an area from the system memory (UPOL0 or UPOL1) (mem during configuration)
- Secure a memory pool from the system memory (UPOL0 or UPOL1) (during configuration or when cre_mpl is issued)
- Acquire a memory block from the memory pool (by issuing get_blk, pget_blk, or tget_blk)

The capacity of each memory area (system memory, memory pool, memory block) must be obtained by adding 8 bytes to the size actually used by the application (for memory area management), and aligning the value to 4 bytes.

Table C-8 Size of Memory Pool

Object Name	Method of Calculating Size (in bytes)
Memory pool	align 4 (Size of memory pool + 8) The value of the memory pool size is the same as the value at the time of the issuance of cre_mpl or the value of Memory pool information *mpl_siz*.

C.6 Examples of Estimating Memory Capacity

This section shows examples of estimating the capacity of the memory area used as the management area (SPOL and UPOL) of the RX850 Pro. In these examples, it is assumed that the V850E1 core is used as the CPU, and that the system calls "cre_tsk" and "cre_mpl" are not issued.

< Application information >

Information	Value (in bytes)
Stack area for interrupt handler *intstk_siz*	256 bytes from SPOL0
Task priority range pri_lvl	15
Maximum number of tasks *maxtsk*	2
Maximum number of semaphores *maxsem*	1
Maximum number of eventflags *maxflg*	2
Maximum number of mailboxes *maxmbx*	3
Maximum number of interrupt handlers *maxint*	4
Maximum number of memory pools *maxmpl*	2
Maximum number of cyclic handlers *maxcyc*	1
Maximum number of extended SVC handlers *maxsvc*	1
Maximum interrupt source number *maxintfactor*	56
Task stack information	256 bytes from SPOL0 0 byte from SPOL1
Temory pool information	4096 bytes from UPOL0 8192 bytes from UPOL1

< Estimation method] >

Object Information	Calculation Expression Size (in bytes)
Operating system management table	[from SPOL0] 504 + align 32 (15 + 4) / 8 + align 4 ((15 + 4) * 2) = 548
System memory management block	[from SPOL0] 8 * 4 = 32
Task management block	[from SPOL0] 56 * 2 = 112
Semaphore management block	[from SPOL0] 20 * 1 = 20
Eventflag management bloc	[from SPOL0] 20 * 2 = 40
Mailbox management block	[from SPOL0] 20 * 3 = 60
Interrupt handler management block	[from SPOL0] 16 * 4 + align 4 (56) = 120
Memory pool management block	[from SPOL0] 24 * 2 = 48
Cyclic handler management block	[from SPOL0] 40 * 1 = 40
Extended SVC handler management block	[from SPOL0] 16 * 1 = 16
Task stack	[from SPOL0] align 4 (28 + 148 + 256) + align 4 (28 + 148 + 256) = 864
Interrupt handler stack	[from SPOL0] align 4 (144 + 28 + 256) = 428
Memory pool	[from UPOL0] 4096 + 8 = 4104 [from UPOL1] 8192 + 8 = 8200

From the above calculation result, the following capacity is necessary.

SPOL0: 548 + 32 + 112 + 20 + 40 + 60 + 120 + 48 + 40 + 16 + 864 + 428 = 2328 bytes
SPOL1: 0 bytes
UPOL0: 4104 bytes
UPOL1: 8200 bytes

APPENDIX D INDEX

[MEMO]

For further information, please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
http://www.cn.necel.com/

Shanghai Branch
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai, P.R.China P.C:200120
Tel:021-5888-5400
http://www.cn.necel.com/

Shenzhen Branch
Unit 01, 39/F, Excellence Times Square Building,
No. 4068 Yi Tian Road, Futian District, Shenzhen,
P.R.China P.C:518048
Tel:0755-8282-9800
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
http://www.tw.necel.com/

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
http://www.kr.necel.com/

G0706