User's Manual

RV850

Real-Time Operating System

User's Manual: Functionality

Target Device

RH850 Family (RH850G3K)

RH850 Family (RH850G3M)

RH850 Family (RH850G3KH)

RH850 Family (RH850G3MH)

Renesas Electronics
www.renesas.com

Rev.1.03 2015.12

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

How to Use This Manual

Readers

This manual is intended for users who design and develop application systems using RH850 family microcontrollers.

Purpose

This manual is intended for users to understand the functions of real-time OS "RV850" manufactured by Renesas Electronics, described the organization listed below.

Organization

This manual consists of the following major units.

1.GENERAL INFORMATION
2.BUILDING THE SYSTEM
3.TASK MANAGEMENT
4.INTERRUPT HANDLING
5.RESOURCE MANAGEMENT
6.EVENT MANAGEMENT
7.COUNTER MANAGEMENT
8.ALARM MANAGEMENT
9.SCHEDULE TABLE MANAGEMENT
10.OS-APPLICATION MANAGEMENT
11.OS EXECUTION MANAGEMENT
12.SCHEDULE MANAGEMENT
13.SYSTEM INITIALIZATION
14.SYSTEM SERVICES
A.CONFIGURATOR
B.CF FILES (OIL)
C.MEMORY FOOTPRINT

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 of the RH850 family microcontroller.
-> 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
Remarks:	Supplementary information
Numeric representation:	Decimal ... XXXX
	Hexadecimal ... 0xXXXX

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

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

Related Documents

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

Document Name	Document No.	
	English	Japanese
RV850 Real-Time Operating System User's Manual: Functionality	This manual	R20UT2768J

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

TABLE OF CONTENTS

1. GENERAL INFORMATION 9
1.1 Overview 9
1.2 Features 9
1.3 Structure 9
1.4 Execution Environment 10
1.5 Folder Structure 11
1.5.1 Object release version 11
1.5.2 Source release version 13
2. BUILDING THE SYSTEM 15
2.1 Overview 15
2.2 Writing User-Own Coding Modules 15
2.2.1 Generating user-own libraries 16
2.3 Writing Processing Programs 16
2.4 Writing CF Files 17
2.4.1 Generating information files 17
2.5 Writing the Linker Directive File 17
2.6 Generation of Load Module 18
3. TASK MANAGEMENT 19
3.1 Overview 19
3.1.1 Task states 19
3.1.2 Stack monitoring facilities 20
3.1.3 Tasks 21
3.1.4 Processing in tasks 21
3.1.5 Generation of tasks 21
3.1.6 Termination of tasks 22
3.2 System Services 23
4. INTERRUPT HANDLING 24
4.1 Overview 24
4.1.1 Stack monitoring facilities 24
4.2 Boot Process 24
4.2.1 Processing in boot process 25
4.3 Interrupt Service Routines 26
4.3.1 Processing in interrupt service routines 28
4.3.2 Registration of interrupt service routines 29
4.3.3 Termination of interrupt service routines 29
4.4 System Services 30

4.5 User-Own Coding Modules . . . 30
4.5.1 Entry process (direct branch method exception vector) . . . 30
4.5.2 Exception/interrupt safety measure process . . . 33
4.6 Multiplex Interrupts . . . 33
5. RESOURCE MANAGEMENT . . . 35
5.1 Overview . . . 35
5.1.1 Ceiling values . . . 35
5.1.2 Scheduler resource . . . 35
5.2 Generation of Resources . . . 36
5.3 System Services . . . 36
6. EVENT MANAGEMENT . . . 37
6.1 Overview . . . 37
6.2 Generation of Events . . . 37
6.3 System Services . . . 37
7. COUNTER MANAGEMENT . . . 38
7.1 Overview . . . 38
7.1.1 System counters . . . 38
7.2 Generation of Counters . . . 38
7.3 System Services . . . 38
8. ALARM MANAGEMENT . . . 39
8.1 Overview . . . 39
8.2 Alarm Callback . . . 39
8.2.1 Processing in alarm callbacks . . . 39
8.2.2 Registration of alarm callbacks . . . 40
8.3 Generation of Alarms . . . 40
8.3.1 System Services . . . 40
9. SCHEDULE TABLE MANAGEMENT . . . 41
9.1 Overview . . . 41
9.2 Schedule Tables . . . 41
9.2.1 Schedule table states . . . 41
9.3 Generation of schedule tables . . . 41
9.4 System Services . . . 41
10. OS-APPLICATION MANAGEMENT . . . 42
10.1 Overview . . . 42
10.1.1 Reliability . . . 42
10.1.2 States . . . 42
10.1.3 Memory protection . . . 43

10.1.4 Peripheral I/O protection function . . . 43
10.2 Trusted Functions . . . 45
10.2.1 Processing in trusted functions . . . 45
10.2.2 Registration of trusted functions . . . 46
10.2.3 Inherited data of trusted functions . . . 46
10.3 OS-Application-Specific Hook Routines . . . 48
10.3.1 Processing in OS-Application-specific hook routines . . . 49
10.3.2 Registration of OS-Application-specific hook routines . . . 50
10.4 Generation of OS-Applications . . . 50
10.5 System Services . . . 50

11. OS EXECUTION MANAGEMENT . . . 51
11.1 Overview . . . 51
11.2 Common Hook Routines . . . 51
11.2.1 Processing in common hook routines . . . 54
11.2.2 Registration of common hook routines . . . 56
11.2.3 System Services . . . 56

12. SCHEDULE MANAGEMENT . . . 57
12.1 Overview . . . 57
12.2 Hook Routines . . . 57
12.3 Idle Handler . . . 58
12.3.1 Processing in idle handler . . . 58

13. SYSTEM INITIALIZATION . . . 59
13.1 Overview . . . 59
13.2 Entry Process (Direct Branch Method Exception Vector) . . . 60
13.3 Boot Process . . . 60
13.4 Kernel Initialization Module . . . 60
13.5 Hook Routines . . . 60

14. SYSTEM SERVICES . . . 61
14.1 Overview . . . 61
14.1.1 Calling of system services . . . 62
14.2 Data Macros . . . 63
14.2.1 Data types . . . 63
14.2.2 Error status . . . 65
14.2.3 Invalid task identifier . . . 66
14.2.4 Task states . . . 66
14.2.5 Schedule table states . . . 66
14.2.6 Exit with error (abend) . . . 66
14.2.7 Access privilege types . . . 67
14.2.8 Object types . . . 67

14.2.9 Checking for access privileges . . . 67
14.2.10 Restart options . . . 68
14.2.11 State of OS-Application . . . 68
14.2.12 System service identifiers . . . 69
14.2.13 Counter information . . . 71
14.2.14 Checking for access privileges . . . 72
14.3 Data Structures . . . 73
14.3.1 Alarm base information . . . 73
14.4 System Services Reference . . . 74
14.4.1 Task management . . . 77
14.4.2 Interrupt handling . . . 87
14.4.3 Resource management . . . 98
14.4.4 Event management . . . 103
14.4.5 Counter management . . . 111
14.4.6 Alarm management . . . 118
14.4.7 Schedule table management . . . 129
14.4.8 OS-Application management . . . 141
14.4.9 OS execution management . . . 159
14.4.10 Utility functions . . . 165

A. CONFIGURATOR . . . 174
A.1 Overview . . . 174
A.2 Activation Method . . . 174
A.2.1 Command file . . . 176
A.3 Sample Command Input . . . 177
A.4 Messages . . . 178
A.4.1 Fatal errors . . . 178
A.4.2 Abort errors . . . 182
A.4.3 Warnings . . . 184

B. CF FILES (OIL) . . . 185
B.1 Overview . . . 185
B.2 Configuration Information . . . 186
B.3 Include Files . . . 187
B.4 CPU . . . 188
B.4.1 Alarm information . . . 189
B.4.2 Application mode information . . . 193
B.4.3 OS-Application information . . . 194
B.4.4 Counter information . . . 200
B.4.5 Event information . . . 203
B.4.6 Interrupt service routine information . . . 205
B.4.7 OS information . . . 207
B.4.8 Resource information . . . 212

B.4.9 Schedule table information . . . 214
B.4.10 Task information . . . 219
B.4.11 System information . . . 223

C. MEMORY FOOTPRINT . . . 226
C.1 Overview . . . 226
C.2 Standard Code Area (.kernel_system) . . . 226
C.3 Interface Area (.kernel_interface) . . . 226
C.4 Constant Data Area (.kernel_const) . . . 227
C.5 Constant Data Area (.kernel_identifier) . . . 231
C.6 Variable Data Area (.kernel_work) . . . 233
C.6.1 Priority buffers . . . 235
C.7 Stack Area (.kernel_stack) . . . 236
C.7.1 System stack . . . 237
C.7.2 OS-Application stack . . . 240
C.7.3 Task stack (extended task) . . . 242
C.8 Interrupt Handler Address Table (.kernel_address) . . . 244

Revision Record . . . 245

1. GENERAL INFORMATION

1.1 Overview

The RV850 is real time multitasking OS developed with the aim of providing an efficient real-time process and multitasking environment, and for enlarging the arena of embedded control application running on the target devices.

1.2 Features

The RV850 has the following features.

(1) Compliance with OSEK/VDX and AUTOSAR specifications
It is designed in compliance with the OSEK/VDX specifications (OSEK/VDX Operating System ver. 2.2.3, OSEK/VDX OSEK Implementation Language ver. 2.5) for the industry-standard open-ended architectures used in in-vehicle distributed control systems, and the AUTOSAR specification (AUTOSAR Specification of Operating System Ver.5.0.0 R4.0 Rev 3), and provides various functions.

(2) High portability
The hardware-dependent processes required by the RV850 to execute the processes for supporting various execution environments are extracted as RV850 dependent modules (user-own coding module), and provided as sample source files.
This improves the portability towards various execution environments and facilitates customization as well.

(3) ROMization
It is designed for small footprint and ROMability in embedded environments.
When the scalability class is SC1 in the RV850, only system services used by the user in the application system are linked at system build-time, enabling the development of a compact real-time multitasking environment that best suits the users' needs.

(4) Utilities
The following useful utilities are provided for system development.

- Configurator
  This tool reads the CF file as the input file and outputs the information files (SIT file, ENTRY file, and kernel macro file) in which is saved the Configuration Information provided by the RV850.

1.3 Structure

The RV850 consists of the two modules given below.

(1) RV850 common module
This processing part is the nucleus of RV850 and offers the following functions.

- Allocates area for objects
- Object initialization process
- Responds to system services
- Selection of processing program corresponding to the events

(2) RV850 dependent module
This is a hardware-dependent process required for executing RV850 processes.
Note that since this processing module is dependent on the execution environment of the processing program, it is separated out as a user-own coding module and provided as a sample source file.

1.4 Execution Environment

The execution environment required for executing RV850 processes is as follows:

(1) Devices
The RV850 has the following target devices.

- RH850 family (G3K core, G3M core, G3KH core, G3MH core)

Remark 1. In the RV850, interrupt handling is implemented by using various registers (e.g. PSW, PMR) provided by the device.

Remark 2. In the RV850 supporting scalability class SC3, illegal memory accesses are prevented by using the memory protection function (MPU) provided by the device.

Remark 3. The RV850 does not support the SIMD operation function provided by the device.

(2) Peripheral controllers
In order to support a wide range of execution environments, the RV850 separates out processes dependent on the execution environment as user-own coding modules, provides them as sample source files, and tries to improve the portability.

Remark 1. In the RV850, table reference method is adopted as the mode to select the interrupt handler address.
Therefore, the RV850 does not operate on hardware not equipped with the table reference method.

Remark 2. In the RV850, interrupt handling is implemented by using interrupt controller INTC1 and INTC2.

Remark 3. In the RV850, a safety function working with the SPID bit (system protection identifier) of the machine configuration register (MCFG0) is used to implement peripheral protection which is prescribed in scalability class SC3 (only in G3M core, G3KH core, G3MH core) conforming to the AUTOSAR specifications.
Consequently, the RV850 does not perform peripheral protection in hardware not equipped with the safety function.

The following shows the OS reserved resources which are occupied by the RV850, and manipulating them from processing programs is prohibited.

Table 1.1 OS Reserved Resources Occupied by RV850

Resource Name	Scalability Class
General-purpose register (r2)	SC1 or SC3
SPID bit of machine configuration (MCFG0)	SC3 (only in G3M core, G3KH core, G3MH core)
Base address (INTBP) of interrupt handler address table	SC1 or SC3
SYSCALL operation setting (SCCFG)	SC3
SYSCALL base pointer (SCBP)	SC3
UM bit of program status word (PSW)	SC1 or SC3
Interrupt priority mask (PMR)	SC1 or SC3
Interrupt function setting (INTCFG)	SC1 or SC3
EITB*n* bit and EIP3*n*-0*n* bit of EI level interrupt control register (EIC*n*) of the interrupt controller (INTC1 or INTC2) corresponding to the interrupts which are defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"	SC1 or SC3
Memory protection function (MPU)	SC3

1.5 Folder Structure

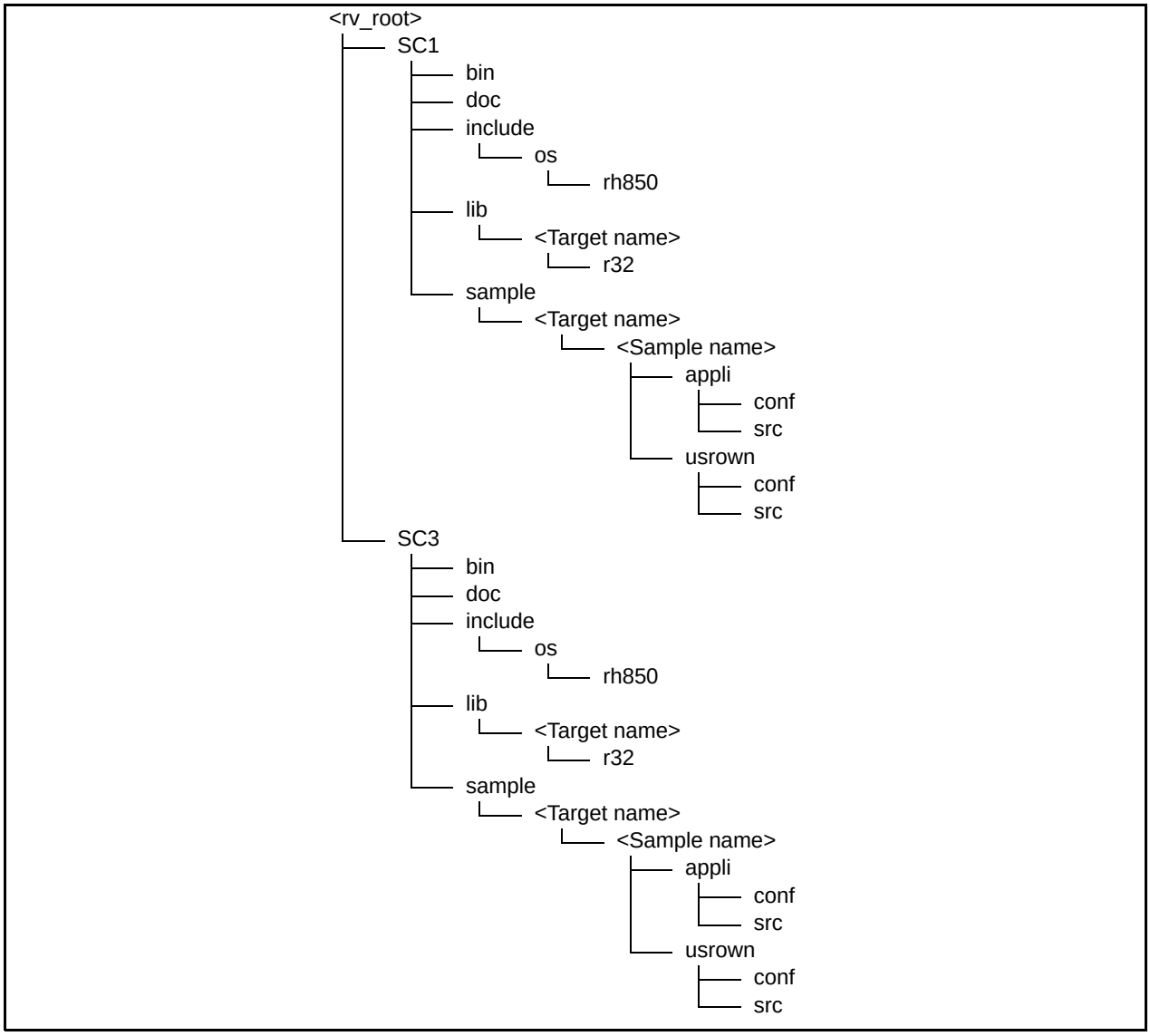
The folder structure of files expanded on the host machine differs according to the file distribution formats provided.

- Object release version
- Source release version

1.5.1 Object release version

The figure below shows the folder structure generated when files provided by the RV850 (object release version) are expanded on the host machine.

Figure 1.1 Folder Structure (Object Release Version)



Details of each folder are shown below.

(1) <rv_root>
The "RV850 installation destination" you specified upon expansion.

(2) <rv_root>\SC1
The RV850 files (scalability class SC1) are stored in this folder.

(3) <rv_root>\SC3
The RV850 files (scalability class SC3) are stored in this folder.

(4) <rv_root>\{SC1, SC3}\bin
The utilities of the RV850 are stored in this folder.
Os_Configurator.exe: Configurator
AUTOSAR_RENESAS_OS_ECUConfigurationParameters.arxml:
OS module definition file (for the configuration editor)

(5) <rv_root>\{SC1, SC3}\doc
The document files of the RV850 are stored in this folder.

(6) <rv_root>\{SC1, SC3}\include
The standard header files of the RV850 are stored in this folder.
Os.h: Standard header file
MemMap.h: AUTOSAR standard header file

(7) <rv_root>\{SC1, SC3}\include\os
The header files of the RV850 are stored in this folder.

(8) <rv_root>\{SC1, SC3}\include\os\rh850
The header files of the RV850 (device dependency: RH850 family) are stored in this folder.

(9) <rv_root>\{SC1, SC3}\lib\<Target name>\r32
The kernel library files (32 register mode) of the RV850 are stored in this folder.
libecc2extsc1.a: ECC2, extended status, SC1, FPU not supported
libecc2extsc1_fpu.a: ECC2, extended status, SC1, FPU supported
libecc2extsc3.a: ECC2, extended status, SC3, FPU not supported (only in G3M, G3KH, G3MH core)
libecc2extsc3_fpu.a: ECC2, extended status, SC3, FPU supported (only in G3M, G3KH, G3MH core)
libecc2extsc3_g3k.a: ECC2, extended status, SC3, FPU not supported (only in G3K core)

(10) <rv_root>\{SC1, SC3}\sample\<Target name>\<Sample name>\appli\conf
The files for generating sample load modules are stored in this folder.

(11) <rv_root>\{SC1, SC3}\sample\<Target name>\<Sample name>\appli\src
The source file and header file of the sample program are stored in this folder.

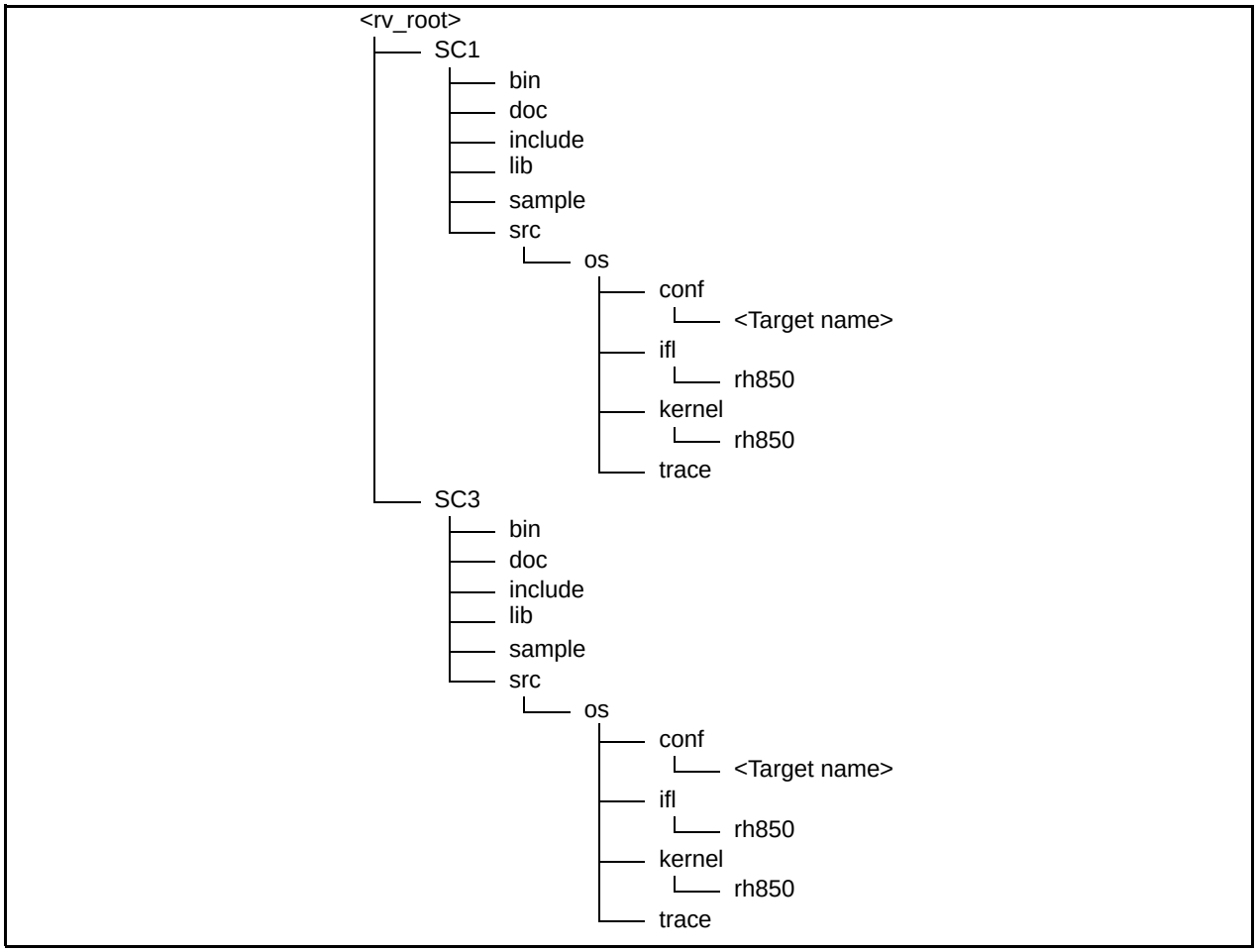
(12) <rv_root>\{SC1, SC3}\sample\<Target name>\<Sample name>\usrown\conf
The user-own library (32 register mode) and files for generating the user-own library (32 register mode) are stored in this folder.
libusr.a: User-own library

(13) <rv_root>\{SC1, SC3}\sample\<Target name>\<Sample name>\usrown\src
The source files of the user-own coding module are stored in this folder.

1.5.2 Source release version

The figure below shows the folder structure generated when files provided by the RV850 (source release version) are expanded on the host machine.

Figure 1.2 Folder Structure (Source Release Version)



Details of each folder are shown below.

(1) <rv_root>
The "RV850 expansion destination" you specified upon expansion.

(2) <rv_root>\SC1
The RV850 files (scalability class SC1) are stored in this folder.

(3) <rv_root>\SC3
The RV850 files (scalability class SC3) are stored in this folder.

(4) <rv_root>\{SC1, SC3}\{bin, doc, include, lib, sample}
This folder has a similar folder configuration and file arrangement as that in "1.5.1Object release version".

(5) <rv_root>\{SC1, SC3}\src\os\conf\<Target name>
The files for generating the kernel library (32 register mode) are stored in this folder.
The kernel library (32 register mode) that is generated using files located in this folder will be stored in the <rv_root>\{SC1, SC3}\lib\<Target name>\r32 folder.

(6) <rv_root>\{SC1, SC3}\src\os\ifl
The interface files of the RV850 are stored in this folder.

(7) <rv_root>\{SC1, SC3}\src\os\ifl\rh850
The interface files of the RV850 (device dependency: RH850 family) are stored in this folder.

(8) <rv_root>\{SC1, SC3}\src\os\kernel
The source files of the kernel library (32 register mode) are stored in this folder.

(9) <rv_root>\{SC1, SC3}\src\os\kernel\rh850
The source files of the kernel library (32 register mode) (device dependency: RH850 family) are stored in this folder.

(10) <rv_root>\{SC1, SC3}\src\os\trace
The source files of the trace routine are stored in this folder.

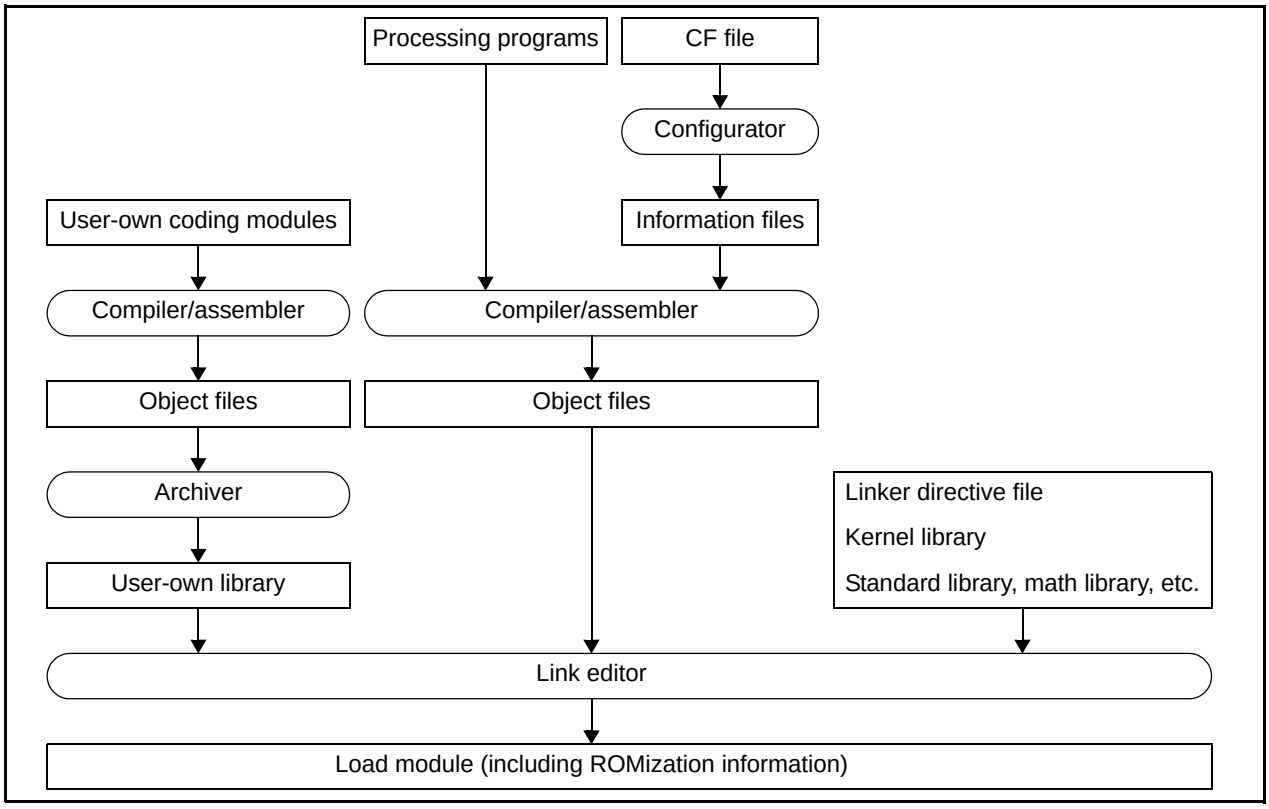
2. BUILDING THE SYSTEM

This chapter describes how to build a system (load module) that uses the functions provided by the RV850.

2.1 Overview

The procedure for building a system is shown below.

Figure 2.1 Procedure for Building System



Remark Since symbol names starting with _kernel or _KERNEL are to be OS reserved symbols in the RV850, usage for a purpose other than the specified purpose is inhibited.

2.2 Writing User-Own Coding Modules

In the RV850, the hardware-dependent processes required by the RV850 to execute the processes for supporting various execution environments are extracted as RV850 dependent modules ("user-own" (user owned) coding modules), and provided as sample source files (direct_vector.850, excent.850). This improves portability to various execution environments and facilitates customization as well.

The following shows a list of "user-own" coding modules extracted by function.

(1) Entry process (direct branch method exception vector)
The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level Interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode", etc.) has been generated.

(2) Exception/interrupt safety measure process
The exception/interrupt safety measure process is a routine dedicated to the safety measure process that is called from entry process when FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated.

Remark See "4.5.1Entry process (direct branch method exception vector)" and "4.5.2Exception/interrupt safety measure process" for details about user-own coding modules.

2.2.1 Generating user-own libraries

Execute the compiler/assembler/archiver for the source files (exception/interrupt safety measure process: excent.850) generated in "2.2Writing User-Own Coding Modules" to generate the library file ("user-own" library).

Remark See the user's manual of your compiler package for details about the compiler/assembler/archiver.

2.3 Writing Processing Programs

Processing programs describe the processing that the system is to carry out.
The RV850 classifies processing programs into the following main types, according to the type of processing they perform.

(1) Boot process
This is a routine dedicated to initialization processing, extracted to initialize the minimum hardware that is required by the RV850 for executing processes.

(2) Task
This is a process routine that is not executed unless the state is manipulated by issuance of a system service, or if conditions defined in a CF file are met.

(3) Interrupt service routine
This is a routine dedicated to the interrupt process that is called when an EI level interrupt (defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode") generates.
The RV850 supports two categories, in consideration of the responsiveness from the occurrence of the interrupt until the interrupt service routine is called.

- Category 1
- Category 2

(4) Alarm callback (only in SC1)
This is a routine dedicated to the expiry action that is called when the alarm has expired.

(5) Trusted function (only in SC3)
For OS-Applications whose reliability has been ensured, it is possible to assign specific trusted functions to individual OS-Applications.

(6) Common hook routine
In the RV850, six types of hook routines with different usages are supported as common hook routines.

- StartupHook
- ShutdownHook
- PostTaskHook
- PreTaskHook
- ErrorHook
- ProtectionHook (only in SC3)

(7) OS-Application-specific hook routine (only in SC3)
Three types of hook routines with different usages are supported as OS-Application-specific hook routines for individual OS-Applications.

- StartupHook_*OsApplication*
- ShutdownHook_*OsApplication*
- ErrorHook_*OsApplication*

(8) Idle handler
This is a routine dedicated to idle processing that is extracted for effectively using the low-power support function provided in target devices.

Remark For details on processing programs, see "3.1.3Tasks", "4.2Boot Process", "4.3Interrupt Service Routines", "8.2Alarm Callback", "10.2Trusted Functions", "11.2Common Hook Routines", "10.3OS-Application-Specific Hook Routines", and "12.3Idle Handler".

2.4 Writing CF Files

Write the CF file required for generating information files (SIT file, ENTRY file, and kernel macro file) that store the data provided for the RV850.

Remark See "B.CF FILES (OIL)" for details about CF files.

2.4.1 Generating information files

Execute the configurator on the CF file wrote in "2.4Writing CF Files" to generate the information files (SIT file, ENTRY file, and kernel macro file).

Remark See "A.CONFIGURATOR" for details about the configurator.

2.5 Writing the Linker Directive File

Write the linker directive file required to fix the memory allocation executed by the link editor.

The allocation location (name, attribute, and allocation destination) of objects modulated in functional units is designated by the RV850. Consequently, in addition to the user-coded program, the provided allocation location must be defined in the linker directive file.

A list of allocation locations provided by the RV850 is shown below.

Table 2.1 Object Allocation Locations Prescribed by RV850

Name	Attribute	Allocation Destination	Description
.kernel_system	AX	ROM	Standard code area
.kernel_interface	AX	ROM	Interface area
.kernel_const	A	ROM	Constant data area (ROM)
.kernel_identifier	A	ROM	Constant data area
.kernel_work	ABW	RAM	Variable data area (RAM)
.kernel_stack	ABW	RAM	Stack area
.kernel_*address*	A	ROM	Interrupt handler address table

Remark 1. The keywords in the "Attribute" column have the following meanings.
- A: Access is possible
- B: No initial value
- W: Write is possible
- X: Execution is possible

Remark 2. Of the s designated by the RV850, .kernel_interface and .kernel_identifier must be allocated to areas that be accesible from all OS-Applications.

Remark 3. The user does not need to code the definition related to .kernel_*address* for the linker directive file, because the configurator outputs to the ENTRY file about the location of the interrupt handler address table .kernel_*address*.
The "*address*" of .kernel_*address* was calculated via "Base address "OsInterruptBaseAddress" + 4 * Interrupt channel number".

Remark 4. See the user's manual of your compiler package for details about the linker directive file.

2.6 Generation of Load Module

Generate the load module (including ROMization information) by executing the compiler/assembler/link editor on the files generated in "2.2Writing User-Own Coding Modules" to "2.5Writing the Linker Directive File", and the library files provided by the RV850 and the compiler package.

List of files necessary at the time of generating the load module (including ROMization information) is given below.

(1) Library files generated in "2.2Writing User-Own Coding Modules"

- User-own library

(2) Source files created in "2.3Writing Processing Programs"

- Processing programs (boot process, tasks, interrupt service routines, alarm callback, trusted functions, common hook routines, OS-Application-specific hook routines, and idle handlers)

(3) Information files created in "2.4Writing CF Files"
Information files (SIT file, ENTRY file, and kernel macro file)

(4) Linker directive files created in "2.5Writing the Linker Directive File"

- Linker directive file

(5) Library files provided by the RV850

- Kernel library

(6) Library files provided by the compiler package

- Standard library, math library, etc.

Remark 1. The RV850 has exclusive ownership of the program register r2. Consequently, the "-reserve_r2" option must be specified in order to compile/assemble a processing program.

Remark 2. The RV850 disables the tiny data area (TDA). Consequently, the "-notda" option must be specified in order to compile/assemble a processing program.

Remark 3. The RV850 does not guarantee correct operation when callt instructions are used. Consequently, the "-no_callt" option must be specified in order to compile/assemble a processing program.

Remark 4. The RV850's standard header file "Os.h" includes the AUTOSAR standard header files "Std_Types.h" and "MemMap.h". Consequently, if you do not need to include these header files, then you must specify the "-D__WITHOSONLY" option when compiling/assembling each file.
When no floating-point operations are used at all (when the -fnone option is specified) in the system to be build, then in addition to the -D__WITHOSONLY__ option, the -D__NOFLOAT__ option must be specified.

Remark 5. When compiling the SIT file (Os_Cfg.c), it is recommended to specify the -sda=0 option.
If -sda=0 is not specified, there is a possibility that the RV850 data which is supposed to be stored in the .kernel_const section and .kernel_work section as shown in table 2.1 is stored in the .sbss section and .rosdata section, respectively.
In particular, when -sda=0 is not specified for an SIT file whose scalability class is SC3, the .sbss section and .rosdata section need to be allocated to an area that cannot be written by all non-trusted OS-Applications.

Remark 6. Registering common hook routines as library routines is prohibited in the RV850.

Remark 7. Hook routines, idle handlers, and user-own libraries need to be linked before the kernel library.

Remark 8. When the RV850 operates on hardware not equipped with the floating-point operation coprocessor (FPU), correct operation cannot be guaranteed for load modules linked with kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a which support FPU.

Remark 9. See the user's manual of your compiler package for details about the compiler/assembler/link editor.

3. TASK MANAGEMENT

This chapter describes the task management functions provided by the RV850.

3.1 Overview

The RV850 provides task management functions as a mechanism for manipulating task states.
The RV850 supports two types of tasks which are required by the OSEK/VDX specifications, according to the purpose of the processing that has to be implemented.

(1) Basic tasks
Tasks to which no events have been assigned (tasks not defined with Event identifier "OsTaskEventRef") can transit to three types of states: SUSPENDED state, READY state, and RUNNING state.

The stacks used while a basic task is running are as follows.

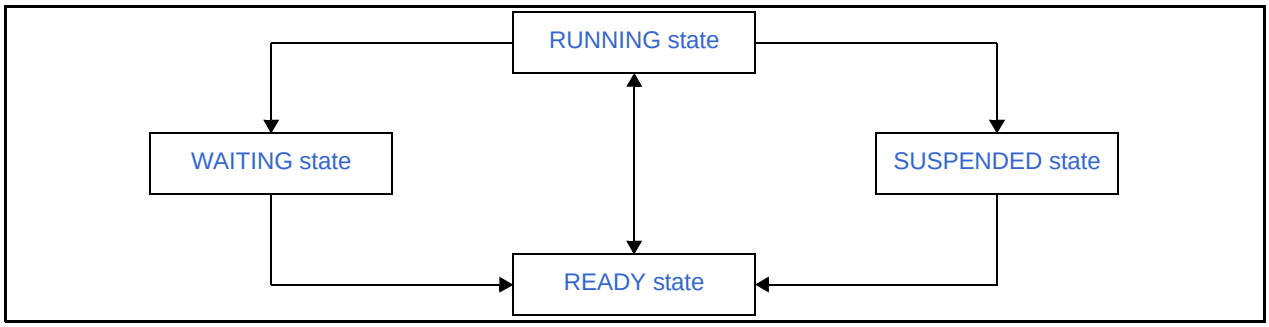
- When the scalability class is SC1
  "System stack" defined in System stack size "OsStackSize"
- When the scalability class is SC3
  "OS-Application stack" defined in OS-Application stack size "OsAppStackSize"

(2) Extended tasks
Tasks to which events have been assigned (tasks defined with Event identifier "OsTaskEventRef") can transit to four types of states: SUSPENDED state, READY state, RUNNING state, and WAITING state.
The stack used while an extended task is running is the "task stack" defined in Task stack size "OsTaskStackSize".

3.1.1 Task states

Tasks change into various states according to the acquisition state of resources required for task execution and the occurrence/non-occurrence of events.
The RV850 classifies and manages the statuses that tasks can take into four main types.

Figure 3.1 Task States



(1) SUSPENDED state
In this state, the task is under the management of the RV850, but it is not subject to scheduling.

(2) READY state
In this state, the task is subject to RV850 scheduling, but it is waiting for allocation of the processor, because another task's process is currently running.

(3) RUNNING state
State in which the processor have been allocated to the task, and is currently being processed.
Only one task can be in the running state at one time in the whole system.

(4) WAITING state
Tasks transition to this state when the conditions required to continue processing as a task are not met.

3.1.2 Stack monitoring facilities

The RV850 provides stack monitoring facilities for detecting stack overflows. This facility checks whether there is enough remaining amount of stacks (task stack, system stack, and OS-Application stack) needed for RV850 processing when control transfers to RV850 processing from task execution. The remaining amount of stacks will be checked at the following timing.

- When starting to process a system service issued from a task
- When starting pre-processing of a category 2 interrupt service routine generated during task execution

If a stack overflow is detected during task execution, this function issues ShutdownOS when the scalability class is SC1 or executes a process (calls the common hook routine ProtectionHook or issues ShutdownOS) according to the definitions of ProtectionHook "OsProtectionHook" when the scalability class is SC3.

Remark 1. Stack monitoring facilities "OsStackMonitoring" is used to enable or disable usage of stack monitoring facilities.

Remark 2. The stack monitoring facilities of the RV850 cannot detect a stack overflow unless control transfers from a task to RV850 processing. If a stack overflow that could occur at a desired timing during task execution needs to be detected, use the Memory protection facility provided by non-trusted OS-Applications of scalability class SC3.

3.1.3 Tasks

Tasks are processing routines that are not executed unless the state is manipulated using a system service, or if conditions defined in a CF file are met.

The basic form for coding a task in the C language is shown below.

```
TASK ( OsTask ) {
     . . . . . . . . . . . . . . . .
     . . . . . . . . . . . . . . . .
}
```

Remark *OsTask* is the task identifier defined in Identifier "OsTask".

3.1.4 Processing in tasks

The RV850 executes its own unique scheduling process when switching tasks. Consequently, note the following points when coding tasks.

(1) Saving/Restoring registers
In the RV850, when switching tasks, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
In the RV850, when activating a task, the contents of the floating-point configuration/status register (FPSR) are changed to the value defined in FPSR "OsAppDefaultFPSRValue" or FPSR default value "OsDefaultFPSRValue". Consequently, it is not necessary to code the save/restore processes of FPSR when TRUE is defined in FPSR saving/restoring "OsSaveFpuReg".

Remark 1. The save/restore processes of FPSR are executed only when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.

Remark 2. If the task executes floating-point operations using imprecise exception and is needed to complete the operations before RV850 restores FPSR, it is necessary to issue syncp and synce operation just before the end of task (before issuing TerminateTask and ChainTask).

(3) Stack switching
In the RV850, when switching tasks, it switches to the task stack defined in Task stack size "OsTaskStackSize", the system stack defined in System stack size "OsStackSize", or the OS-Application stack defined in OS-Application Stack size "OsAppStackSize".
Consequently, it is not necessary to write code to switch the stack.

Remark Switching to the system stack defined in System stack size "OsStackSize" is carried out only when the new task after switching is a basic task (SC1).
Switching to the OS-application stack defined in OS-Application Stack size "OsAppStackSize" is possible only when the new task after switching is a basic task (SC3).

(4) Interrupt acceptance
In the RV850, when switching tasks, operations related to interrupt acceptance are not carried out.
Consequently, it is necessary to code a process to issue EnableAllInterrupts, DisableAllInterrupts, etc. to explicitly change the acceptance status of interrupts.

(5) Issuing system services
Only system services that are allowed to be issued from tasks are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

3.1.5 Generation of tasks

The RV850 restricts task generation to static generation via the definition of Task information. Consequently, tasks cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
A task is statically generated by defining Task information in a CF file.
In the RV850, the kernel initialization module reads the Task information definitions from the information file, and initializes these tasks, which are subject to management.

3.1.6 Termination of tasks

A task is ended (shifted to SUSPENDED state) by issuing TerminateTask or ChainTask.

In the RV850, a task can end without issuing TerminateTask or ChainTask, and the following operations are performed in this case.

[For scalability class SC1]

(1) If the task has acquired any internal resources, the internal resources will be released.

(2) If the task has acquired any internal resources, the current priority will be changed (the current priority of the task is returned to Initial priority "OsTaskPriority").

(3) If the common hook routine (PostTaskHook) is registered, PostTaskHook will be called.

(4) The task transits from RUNNING state to SUSPENDED state.

(5) The task is unlinked from the ready queue corresponding to the priority.

(6) If the task is a basic task, the activation request counter will be decremented (0x1 is subtracted from the activation request counter).

(7) If the subtraction result of the activation request counter is greater than 0x0, the task transits from SUSPENDED state to READY state.

(8) The scheduler is activated.

[For scalability class SC3]

(1) If the task has issued DisableAllInterrupts, SuspendAllInterrupts, or SuspendOSInterrupts but has not yet issued the corresponding EnableAllInterrupts, ResumeAllInterrupts, or ResumeOSInterrupts, the process to enable the acceptance of interrupts will be performed (process equal to the corresponding EnableAllInterrupts, ResumeAllInterrupts, or ResumeOSInterrupts).

(2) If the task has acquired any normal resources, the normal resources will be released.

(3) If the task has acquired any normal resources, the current priority will be changed (the current priority of the task is returned to Initial priority "OsTaskPriority").

(4) If the task has acquired any internal resources, the internal resources will be released.

(5) If the task has acquired any internal resources, the current priority will be changed (the current priority of the task is returned to Initial priority "OsTaskPriority").

(6) When the common hook routine (ErrorHook) is registered, if the task has issued DisableAllInterrupts, SuspendAllInterrupts, or SuspendOSInterrupts but has not yet issued the corresponding EnableAllInterrupts, ResumeAllInterrupts, or ResumeOSInterrupts, or if the task has acquired any normal resources, ErrorHook is called with E_OS_MISSINGEND (0x14) as the parameter.

(7) When an OS-Application-specific hook routine (ErrorHook_*OsApplication*) is registered in the OS-Application to which the task belongs, if the task has issued DisableAllInterrupts, SuspendAllInterrupts, or SuspendOSInterrupts but has not yet issued the corresponding EnableAllInterrupts, ResumeAllInterrupts, or ResumeOSInterrupts, or if the task has acquired any normal resources, ErrorHook_*OsApplication* is called with E_OS_MISSINGEND (0x14) as the parameter.

(8) If the common hook routine (PostTaskHook) is registered, PostTaskHook will be called.

(9) The task transits from RUNNING state to SUSPENDED state.

(10) The task is unlinked from the ready queue corresponding to the priority.

(11) If the task is a basic task, the activation request counter will be decremented (0x1 is subtracted from the activation request counter).

(12) If the subtraction result of the activation request counter is greater than 0x0, the task transits from SUSPENDED state to READY state.

(13) The scheduler is activated.

Remark 1. In the RV850, if the scalability class is SC1, correct operation cannot be guaranteed when a task that has acquired normal resources is ended without issuing TerminateTask or ChainTask.

Remark 2. In the RV850, if the scalability class is SC1, correct operation cannot be guaranteed when a task that has issued DisableAllInterrupts, SuspendAllInterrupts, or SuspendOSInterrupts but has not yet issued the

corresponding EnableAllInterrupts, ResumeAllInterrupts, or ResumeOSInterrupts is ended without issuing TerminateTask or ChainTask.

Remark 3. The AUTOSAR specifications do not specify the operations to be performed when SC1 is defined in Scalability class "OsScalabilityClass" and the task is ended without issuing TerminateTask or ChainTask. Accordingly, the operations for the case of SC1 which are shown above are unique to the RV850.

3.2 System Services

The system services shown in "14.4.1Task management" are used to manipulate tasks dynamically from processing programs.

4. INTERRUPT HANDLING

This chapter describes the interrupt handling functions provided by the RV850.

4.1 Overview

In the RV850, among the interrupts generated by the device, only the interrupt service routines corresponding to the EI level interrupts which are defined in Exception code "OsIsrExceptionCode" and Exception code "OsCounterException-Code" are subject to management.

Consequently, the routines (boot process, exception/interrupt safety measure process, etc.) corresponding to the reset (RESET), FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrException-Code" or Exception code "OsCounterExceptionCode"), etc. are special subject to management.

4.1.1 Stack monitoring facilities

The RV850 provides stack monitoring facilities for detecting stack overflows.

This facility checks whether there is enough remaining amount of stacks (system stack and OS-Application stack) needed for RV850 processing upon transition to RV850 processing from execution of an interrupt service routine (category 2). The remaining amount of stacks will be checked at the following timing.

- When starting to process a system service issued from an interrupt service routine (category 2)
- When starting pre-processing of a category 2 interrupt service routine generated during execution of an interrupt service routine (category 2)

If a stack overflow is detected during execution of an interrupt service routine (category 2), this function issues ShutdownOS when the scalability class is SC1 and executes a process (calls the common hook routine ProtectionHook or issues ShutdownOS) according to the definitions of ProtectionHook "OsProtectionHook" when the scalability class is SC3.

Remark 1. In the RV850, stack monitoring facilities are applied for only category 2 interrupt service routines.

Remark 2. Stack monitoring facilities "OsStackMonitoring" is used to enable or disable usage of stack monitoring facilities.

Remark 3. The stack monitoring facilities of the RV850 cannot detect a stack overflow unless control transfers from an interrupt service routine (category 2) to RV850 processing. If a stack overflow that could occur at a desired timing during execution of an interrupt service routine needs to be detected, use the Memory protection facility provided by non-trusted OS-Applications of scalability class SC3.

4.2 Boot Process

This is a routine dedicated to initialization processing, extracted to initialize the minimum hardware that is required by the RV850 for executing processes. It is called from the branch process (Entry process (direct branch method exception vector)) that is assigned to the address of the handler to which the device forcibly transfers control when a hardware reset is generated.

In the boot process, the following to be performed as the minimum hardware initialization processing must be described.

(1) Basic system registers
- The EBV and CU*n* bits of the program status word (PSW)
- The reset vector base address (RBASE)
- The exception handler vector address (EBASE)
- The UIC bit of CPU control (MCTL)

(2) Interrupt function registers (only in G3M core, G3KH core)
- The FPI exception interrupt priority setting (FPIPR)

(3) Interrupt control registers
- The EI level interrupt bind register (EIBD*n*)

(4) The safety function associated with the SPID bit (system protection identifier) of the machine configuration register (MCFG0) (only in G3M core, G3KH core, G3MH core)

(5) Peripheral controllers (e.g. timer)

Remark 1. In the RV850, only in cases where the floating-point configuration/status register (FPSR) is to be manipulated, the CU*n* bit of PSW is manipulated (1 is set to the CU0 bit) and after manipulation of FPSR has completed, the CU*n* bit is returned to the value it had before FPSR manipulation.
Consequently, the RV850 starts operation while inheriting the CU*n* bit value set in the boot process, and even when the CU*n* bit is dynamically manipulated from a processing program, operation is to be continued with the value after change being inherited.

Remark 2. In the RV850, table reference method is adopted as the mode to select the interrupt handler address. Therefore, 0 should be set to the RINT bit of RBASE and the RINT bit of EBASE.

Remark 3. The UIC bit of MCTL needs to be manipulated only when a processing program will issue an EI or DI instruction.

Remark 4. A priority higher than that of category 2 interrupt service routines defined in Interrupt service routine information should be set in the FPIPR bits of FPIPR.

Remark 5. Manipulations regarding the access privilege for I/O areas are required only when SC3 is defined in Scalability class "OsScalabilityClass".

Remark 6. For the SPID bit value "0" of the MCFG0 register, make a setting to enable all accesses to I/O areas. See the user's manual of the target device for details about the SPID bit.

Remark 7. The AUTOSAR specifications have a requirement (OS374) specifying that timers should be initialized using a real-time OS. With the RV850, however, timers must be initialized before StartOS is issued (before the RV850 is started).

The basic form for coding the boot process in the assembly language is shown below.

```
    .tex
    .align  4
    .globl  _boot
_boot:
    ...................
    ...................
```

4.2.1 Processing in boot process

This is a routine dedicated to initialization processing, which is called before the RV850 starts. Consequently, the following points must be noted when coding the boot process.

(1) Saving/Restoring registers
There are no registers that must be saved/restored in order to execute the boot process.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
The floating-point configuration/status register (FPSR) is not set when the boot process is started.
Consequently, it is not necessary to code the save/restore processes of FPSR.

(3) Stack switching
The stack pointer (SP) is not set when the boot process is started.
Consequently, to use the stack dedicated to the boot process, it is necessary to code the SP setup process.

(4) Issuing system services
Only system services that are allowed to be issued from boot process are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

4.3 Interrupt Service Routines

This is a routine dedicated to the interrupt process that is called when an EI level interrupt (defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode") generates.

In the RV850, interrupt service routines are independent from tasks. Consequently, when an interrupt is generated, even the highest priority task is preempted during execution, and control is transferred to the interrupt service routine.

The RV850 supports two types of categories required in consideration of the response from the time of occurrence of the interrupt till the activation of the interrupt service routine.

(1) Category 1

This is a routine dedicated to interrupt processing that is activated without intervening the RV850 when an interrupt is generated.

Category 1 interrupt service routines are called directly from the handler address to which the device forcibly shifts control when an interrupt occurs. For this reason, its responsiveness approaches the limits of the hardware.

The basic form for coding a category 1 interrupt service routine in the C language is shown below.

[#pragma ghs interrupt directive is used]

```
#pragma ghs interrupt

ISR ( OsIsr ) {
    ...................
    ...................
    return;
}
```

[#pragma ghs interrupt (enable) directive is used]

```
#pragma ghs interrupt (enable)

ISR ( OsIsr ) {
    ...................
    ...................
    return;
}
```

The basic form for coding a category 1 interrupt service routine in the assembly language is shown below.

```
    .text
    .align  4
    .globl  _ISROsIsr
_ISROsIsr:
    addi    -16, sp, sp
    st.w    r1, 0[sp]
    ...................
    ...................
    ld.w    0[sp], r1
    addi    16, sp, sp
    eiret
```

Remark Since a category 1 interrupt service routine operates in supervisor mode, the memory protection function (MPU) provided by the device cannot be applied.

(2) Category 2

This is a routine dedicated to interrupt processing that is called after interrupt pre-processing (saving register contents, switching stacks, etc.) is executed by the RV850 when an interrupt is generated.

Category 2 interrupt service routines are called from the handler address to which the device forcibly shifts control when an interrupt occurs, via the interrupt pre-processing provided by the RV850. This simplifies processing in the interrupt service routine.

The basic form for coding a category 2 interrupt service routine in the C language is shown below.

```
ISR ( OsIsr ) {
    ...................
    ...................
    return;
}
```

Remark The identifier of the interrupt service routine defined in Identifier "OsIsr" is written in *OsIsr*.

4.3.1 Processing in interrupt service routines

The RV850 supports two categories, in consideration of the responsiveness from the occurrence of the interrupt until the interrupt service routine is activated. Consequently, it is necessary to consider the following points in accordance with the category when coding interrupt service routines.

(1) Category 1

(a) Saving/Restoring registers
The RV850 is not involved with the calling of category 1 interrupt service routines.
Consequently, it is necessary to code the save/restore processes of registers, in accordance with the processing contents.

(b) Saving/Restoring FPSR
The RV850 is not involved with the calling of category 1 interrupt service routines.
Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

Remark If the category 1 interrupt service routine executes floating-point operations using imprecise exception and is needed to complete the operations before transiting into RV850 execution, it is necessary to issue syncp and synce operation just before the end of the category 1 interrupt service routine.

(c) Stack switching
The RV850 is not involved with the calling of category 1 interrupt service routines.
Consequently, it is necessary to code the stack pointer (SP) setting process when using a dedicated stack for category 1 interrupt service routines.

(d) Interrupt acceptance
The RV850 is not involved with the calling of category 1 interrupt service routines.
Therefore, when a category 1 interrupt occurs, the device manipulates the ID bit in the program status word (PSW) to disable the acceptance of interrupts.
Consequently, it is necessary to code the issuance processes, such as EnableAllInterrupts and DisableAllInterrupts, in order to change the status of interrupt acceptance explicitly.

(e) Issuing system services
Only system services that are allowed to be issued from interrupt service routines are issuable.

Remark 1. The RV850 does not guarantee correct operation when an unallowable system service is issued from an interrupt service routine.

Remark 2. See "14.4System Services Reference" for details about the issue scope of each system service.

(2) Category 2

(a) Saving/Restoring registers
When the RV850 transfers control to a category 2 interrupt service routine, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(b) Saving/Restoring FPSR
When the RV850 transfers control to a category 2 interrupt service routine, the contents of the floating-point configuration/status register (FPSR) are changed to the value defined in FPSR "OsAppDefaultFPSRValue" or FPSR default value "OsDefaultFPSRValue".
Consequently, it is not necessary to code the save/restore processes of FPSR when TRUE is defined in FPSR saving/restoring "OsSaveFpuReg".

Remark 1. The save/restore processes of FPSR are executed only when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.

Remark 2. If the category 2 interrupt service routine executes floating-point operations using imprecise exception and is needed to complete the operations before tRV850 restores FPSR, it is necessary to issue syncp and synce operation just before the end of the category 2 interrupt service routine.

(c) Stack switching
When the RV850 transfers control to a category 2 interrupt service routine, it switches to the system stack defined in System stack size "OsStackSize" or the OS-applicatin stack defined in OS-Application Stack size "OsAppStackSize".
Consequently, it is not necessary to write code to switch the stack.

(d) Interrupt acceptance
When the RV850 transfers control to a category 2 interrupt service routine, the acceptance status of interrupts is changed to enabled (ID bit of PSW is manipulated).
Consequently, it is necessary to code the issuance processes, such as DisableAllInterrupts and SuspendAllInterrupts, in order to change the status of interrupt acceptance explicitly.

(e) Issuing system services
Only interrupt service routines or system services that are allowed to be issued from interrupt service routines (category 2) are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

4.3.2 Registration of interrupt service routines

The RV850 restricts registration of interrupt service routines to static registration via the definition of Interrupt service routine information. Consequently, interrupt service routines cannot be registered dynamically, for example by issuing system services from a processing program.

(1) Static registration
An interrupt service routine is statically registered by defining Interrupt service routine information in a CF file.
In the RV850, the kernel initialization module reads the Interrupt service routine information definitions from the information file, and initializes these interrupt service routines, which are subject to management.

In the RV850, among the interrupts generated by the device, only the interrupt service routines corresponding to the EI level interrupts which are defined in Exception code "OsIsrExceptionCode" and Exception code "OsCounterExceptionCode" are subject to management.
Consequently, using a #pragma directive provided by the compiler, routines dedicated to interrupt processing of interrupts other than EI level interrupts should be registered as processing programs not managed by the RV850.

4.3.3 Termination of interrupt service routines

An interrupt service routine is ended (control is returned to the processing program in which an interrupt was generated) by issuing the return instruction (the eiret instruction for the assembly language).
In the RV850, the following operations are performed when an interrupt service routine has issued the return instruction.

(1) For category 1 and scalability class SC1 or SC3

- Control is returned to the processing program in which an interrupt was generated.

(2) For category 2 and scalability class SC1

- Work registers are restored.
- Stacks are switched.
- The floating-point configuration/status register (FPSR) is restored.
- The scheduler is activated.

(3) For category 2 and scalability class SC3

- Work registers are restored.
- Stacks are switched.
- The floating-point configuration/status register (FPSR) is restored.
- If the interrupt service routine has issued DisableAllInterrupts or SuspendAllInterrupts but has not yet issued the corresponding EnableAllInterrupts or ResumeAllInterrupts, the process to enable the acceptance of interrupts will be performed (process equal to the corresponding EnableAllInterrupts or ResumeAllInterrupts).
- When an OS-Application-specific hook routine (ErrorHook_*OsApplication*) is registered in the OS-Application to which the interrupt service routine belongs, if the interrupt service routine has issued DisableAllInterrupts or SuspendAllInterrupts but has not yet issued the corresponding EnableAllInterrupts or ResumeAllInterrupts, ErrorHook_*OsApplication* is called with E_OS_DISABLEDINT (0x15) as the parameter.
- When the common hook routine (ErrorHook) is registered, if the interrupt service routine has issued DisableAllInterrupts or SuspendAllInterrupts but has not yet issued the corresponding EnableAllInterrupts or ResumeAllInterrupts, ErrorHook is called with E_OS_DISABLEDINT (0x15) as the parameter.

- If the interrupt service routine has acquired any normal resources, the normal resources will be released.
- If the interrupt service routine has acquired any normal resources, the current priority will be changed (the current priority of the interrupt service routine is returned to Initial priority "OsIsrPriority").
- If the interrupt service routine has acquired any normal resources, the process to enable the acceptance of interrupts will be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is enabled).
- When an OS-Application-specific hook routine (ErrorHook_*OsApplication*) is registered in the OS-Application to which the task belongs, if the interrupt service routine has acquired any normal resources, ErrorHook_*OsApplication* is called with E_OS_RESOURCE (0x6) as the parameter.
- When the common hook routine (ErrorHook) is registered, if the interrupt service routine has acquired any normal resources, ErrorHook is called with E_OS_RESOURCE (0x6) as the parameter.
- The scheduler is activated.

Remark 1. In the RV850, for category 1, correct operation cannot be guaranteed when an interrupt service routine that has issued DisableAllInterrupts or SuspendAllInterrupts but has not yet issued the corresponding EnableAllInterrupts or ResumeAllInterrupts has issued the return instruction.

Remark 2. The save/restore processes of FPSR are executed only when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.

4.4 System Services

The system services shown in "14.4.2Interrupt handling" are used to manipulate interrupts dynamically from processing programs.

4.5 User-Own Coding Modules

In the RV850, the hardware-dependent processes required by the RV850 to execute the processes for supporting various execution environments are extracted as RV850 dependent modules ("user-own" (user owned) coding modules), and provided as sample source files.

This improves portability to various execution environments and facilitates customization as well.

4.5.1 Entry process (direct branch method exception vector)

The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated.

The basic form for coding the entry process in the assembly language is shown below.

[SC1, RBASE/EBASE: 0x0]

```
    .globl  _entry0000
    .globl  _entry0010
    .........
    .globl  _entry0100
    .........

    .org    0x00000000
_entry0000:
    jr      _boot

    .org    0x00000010
_entry0010:
    jr      kernel_e_IllegalExcEntry

    .........

    .org    0x00000100
_entry0100:
    jr      kernel_e_IllegalExcEntry

    .........
```

[SC3, RBASE/EBASE: 0x0]

```
    .globl  _entry0000
    .globl  _entry0010
    .globl  _entry0020
    .........
    .globl  _entry0090
    .globl  _entry00A0
    .globl  _entry00B0
    .........
    .globl  _entry0100
    .........

    .org    0x00000000
_entry0000:
    jr      _boot

    .org    0x00000010
_entry0010:
    jr      kernel_e_ProtectEntry

    .org    0x00000020
_entry0020:
    jr      kernel_e_IllegalExcEntry

    .........

    .org    0x00000090
_entry0090:
    jr      kernel_e_ProtectEntry

    .org    0x000000A0
_entry00A0:
    jr      kernel_e_ProtectEntry

    .org    0x000000B0
_entry00B0:
    jr      kernel_e_IllegalExcEntry

    .........

    .org    0x00000100
_entry0100:
    jr      kernel_e_IllegalExcEntry

    .........
```

Remark 1. The user does not need to code the entry process associated with EI level interrupts that are defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode", because the configurator outputs to the ENTRY file about the entry process.

Remark 2. If the scalability class is SC3, code the branch process to process _kernel_e_ProtectEntry which corresponds to the protection exception for a system error exception (+0x10 address), memory protection exception (+0x90 address), or privilege instruction exception (+0xA0 address).

4.5.2 Exception/interrupt safety measure process

The exception/interrupt safety measure process is a routine dedicated to the safety measure process that is called from entry process when FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated.

The basic form for coding an exception/interrupt safety measure process in the C language is shown below.

```
#pragma ghs interrupt

void
_kernel_e_IllegalExcEntry ( void ) {
    ...................
    ...................
}
```

The following points should be noted when coding an exception/interrupt safety measure process.

(1) Saving/Restoring registers
The RV850 is not involved with the calling of an exception/interrupt safety measure process routine. Consequently, it is necessary to code the save/restore processes of registers, in accordance with the processing contents of the routine.

(2) Saving/Restoring FPSR
The RV850 is not involved with the calling of an exception/interrupt safety measure process routine. Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

(3) Stack switching
The RV850 is not involved with the calling of an exception/interrupt safety measure process routine. Consequently, it is necessary to code the stack pointer (SP) setting process when using a dedicated stack for the exception/interrupt safety measure process routine.

(4) Interrupt acceptance
When an exception/interrupt safety measure process routine is called, the device manipulates the ID bit in the program status word (PSW) to disable the acceptance of interrupts.

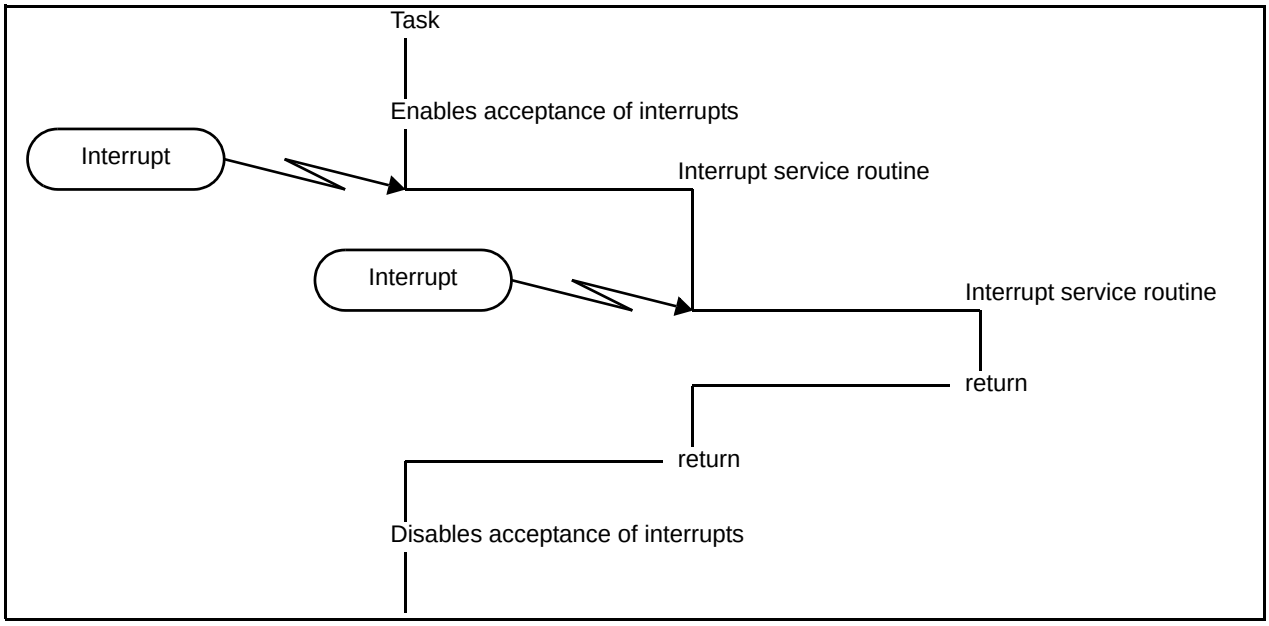
(5) Issuing system services
System services cannot be called from an exception/interrupt safety measure process routine.

Remark An exception/interrupt safety measure process is prepared in the RV850.
Consequently, event _kernel_e_IllegalExcEntry is not coded, if the branch process to the exception/interrupt safety measure process was assigned to entry process, and, in the case of the operand of SYSCALL instruction is an illegal value, the exception/interrupt safety measure process (the process to issue ShutdownOS in which E_OS_SYS_ILLEGAL_EXCEPTION has been specified for parameter *Error*) will be called.

4.6 Multiplex Interrupts

In the RV850, interrupts that are generated more than once during processing of the interrupt service routine are called "multiplex interrupts".

Figure 4.1 Multiplex Interrupts



Remark Since a priority (including Priority "OsCounterPriority" that is specified for the hardware counter) higher than that of any category 2 interrupt service routine is specified in Initial priority "OsIsrPriority" of the category 1 interrupt service routine, the acceptable interrupts during processing of the category 1 interrupt service routine are only category 1 interrupts.

5. RESOURCE MANAGEMENT

This chapter describes the resource management functions provided by the RV850.

5.1 Overview

The RV850 provides resource management facilities as a mechanism for achieving mutual exclusion. Resources are exclusively controlled using "priority ceiling protocols", and racing over resources by tasks or interrupt service routines (category 2) that use the limited number of shared resources (data, peripheral devices, common functions, etc.) or deadlocks can be prevented.

The RV850 supports three types of resource.

(1) Normal resources
These resources can be acquired and released dynamically by issuing GetResource and ReleaseResource.

(2) Internal resources
These resources are allocated automatically by the RV850 when the task state changes from READY to RUNNING.

(3) Linked resources
These resources inherit properties (ceiling values and OS-Application identifiers) from another resource.

5.1.1 Ceiling values

A ceiling value is a priority assigned to the interval between the acquisition and release of a resource by a processing program (task or interrupt service routine).

Consequently, if a task with priority 3 acquires a resource with a ceiling value of 10, then it will run as a priority-10 task until that resource is released, and therefore control will not be transferred to tasks with priority 0 to 10.

If a task with priority 3 acquires a resource with a ceiling value of INTPRI5, then it will operate at the maximum priority of 29 until that resource is released, and in addition, the acceptance of INTPRI0 to INTPRI5 interrupts of will be disabled. For this reason, control will not be passed to tasks with priority 0 to 29, or INTPRI0 to INTPRI5 interrupt service routines.

Remark Interrupts that were put on hold while a resource with a ceiling value of INTPRI0 to INTPRI5 was acquired are accepted after that resource is released.

5.1.2 Scheduler resource

The OSEK/VDX specifications have a definition of a resource, which have a identifier "RES_SCHEDULER" and the ceiling value “29” as a means to dynamically enable and disable the activation of the scheduler from tasks. See "5.2Generation of Resources" for how to apply a scheduler resource RES_SCHEDULER on RV850.

This disables the activation of the scheduler from the time the system service is issued via issuance of GetResource by the task (parameter *ResID* set to RES_SCHEDULER), until ReleaseResource is issued (parameter *ResID* set to RES_SCHEDULER).

Remark 1. The RV850 does not define any scheduler resource automatically to conform to the AUTOSAR specifications. See "5.2 Generation of Resources" for detail about the way to enable the scheduler resource "RES_SCHEDULER".

Remark 2. The ceiling value of the scheduler resource is 29 and it cannot be used for exclusive control related to interrupt service routines.

5.2 Generation of Resources

The RV850 restricts resource generation to static generation via the definition of Resource information. Consequently, resources cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
A resource is statically generated by defining Resource information in a CF file.
In the RV850, the kernel initialization module reads the Resource information definitions from the information file, and initializes these resources, which are subject to management.

Remark The OSEK/VDX specifications have a rule specifying a scheduler resource to be automatically generated by defining TRUE in Scheduler resource "OsUseResScheduler". The AUTOSAR specifications, however, have a rule specifying not to perform automatic generation.
Therefore, when generating a scheduler resource in the RV850, make the following definitions in the CF file.

- Specify RES_SCHEDULER in Identifier "OsResource" of the Resource information.
- Specify 29 in Ceiling value "OsResourcePriority" of the Resource information.

5.3 System Services

The system services shown in "14.4.3Resource management" are used to manipulate resources dynamically from processing programs.

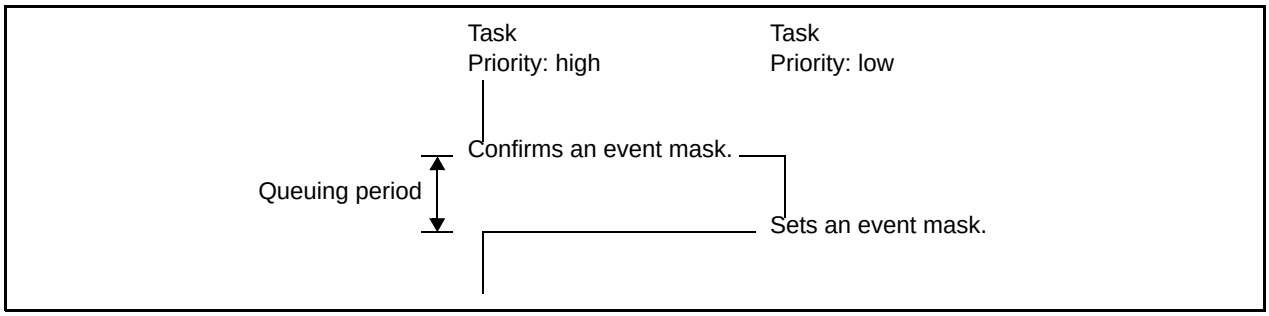
6. EVENT MANAGEMENT

This chapter describes the event management functions provided by the RV850.

6.1 Overview

The RV850 provides event management functions as a mechanism for queuing tasks that are waiting to execute a process until another task outputs the results of its processing.

Figure 6.1 Waiting between Tasks



Remark In the RV850, the distinction between a basic task and an extended task is based on whether an event has been assigned (whether Event identifier "OsTaskEventRef" has been defined).
In the RV850, the maximum number of events that can be assigned to an extended task is 32.

6.2 Generation of Events

The RV850 restricts event generation to static generation via the definition of Event information. Consequently, events cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
An event is statically generated by defining Event information in a CF file.
In the RV850, the kernel initialization module reads the Event information definitions from the information file, and initializes these events, which are subject to management.

6.3 System Services

The system services shown in "14.4.4Event management" are used to manipulate events dynamically from processing programs.

7. COUNTER MANAGEMENT

This chapter describes the counter management functions provided by the RV850.

7.1 Overview

The RV850 provides counter management functions as a mechanism to perform processing in accordance with the number of times an application specific trigger has occurred, and supports two types of counter.

(1) Software counters
The count value is updated when the system service "IncrementCounter" is called from a processing program.

(2) Hardware counters
The count value is updated when an interrupt occurs corresponding to the Exception code "OsCounterExceptionCode" defined in the Counter information.

7.1.1 System counters

Although the RV850 supports the system counter required by the OSEK/VDX specifications, it does not create any distinctions from the ways of handling and using ordinary counters, other than defining the SYS_COUNTER keyword prescribed in Identifier "OsCounter".

7.2 Generation of Counters

The RV850 restricts counter generation to static generation via the definition of Counter information. Consequently, counters cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
A counter is statically generated by defining Counter information in a CF file.
In the RV850, the kernel initialization module reads the Counter information definitions from the information file, and initializes these counters, which are subject to management.

Remark If the target counter is a hardware counter, the RV850 will also register the interrupt service routine (category 2) for updating the count value, as part of the generation process.

7.3 System Services

The system services shown in "14.4.5Counter management" are used to manipulate counters dynamically from processing programs.

8. ALARM MANAGEMENT

This chapter describes the alarm management functions provided by the RV850.

8.1 Overview

The RV850 provides alarm management functions as a mechanism to perform processing in synchronization with the change of counter values.

The alarm management functions perform the expiry action when the count value of a counter associated in Counter identifier "OsAlarmCounterRef" reaches the expiry conditions (expiry count value or cyclic count value). This expiry action includes activating tasks, setting event masks, updating count values, and calling an alarm callback (only in SC1).

8.2 Alarm Callback

This is a routine dedicated to the expiry action that is called when the alarm has expired.

In the RV850, alarm callbacks are independent from tasks. Consequently, when an alarm has expired, even the highest priority task is preempted during execution, and control is transferred to the alarm callback.

The basic form for coding the alarm callback in the C language is shown below.

```
ALARMCALLBACK ( OsAlarmCallbackName ) {
    ..................
    ..................
    return;
}
```

Remark The identifier of the alarm callback defined in Alarm information is set in *OsAlarmCallbackName*.

8.2.1 Processing in alarm callbacks

When the RV850 transfers control from a processing program to an alarm callback, it performs independent pre-processing. It also performs independent post-processing before returning control from an alarm callback to a processing program. Consequently, the following points should be noted when coding an alarm callback.

(1) Saving/Restoring registers
When the RV850 transfers control to the alarm callback, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
When the RV850 transfers control to the alarm callback, the save/restore processes of the floating-point configuration/status register (FPSR) are not executed.
Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

Remark If the alarm callback executes floating-point operations using imprecise exception and is needed to complete the operations before transiting into RV850 execution, it is necessary to issue syncp and synce operation just before the end of the alarm callback.

(3) Stack switching
When the RV850 transfers control to the alarm callback, it switches to the system stack defined in System stack size "OsStackSize".

(4) Interrupt acceptance
When the RV850 transfers control to the alarm callback, the acceptance status of category 2 interrupts is changed to disabled (PM*n* bits of PMR are manipulated).

(5) Issuing system services
Only system services that are allowed to be issued from alarm callbacks are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

8.2.2 Registration of alarm callbacks

The RV850 restricts registration of alarm callbacks to static registration via the definition of Alarm callback identifier "OsAlarmCallbackName". Consequently, alarm callbacks cannot be registered dynamically, for example by issuing system services from a processing program.

(1) Static registration
An alarm callback is statically registered by defining Alarm callback identifier "OsAlarmCallbackName".
In the RV850, the kernel initialization module reads the Alarm callback identifier "OsAlarmCallbackName" definitions from the information file, and initializes these alarm callbacks, which are subject to management.

8.3 Generation of Alarms

The RV850 restricts alarm generation to static generation via the definition of Alarm information. Consequently, alarms cannot be generated dynamically, for example by issuing system services from a processing program.

- Static generation
An alarm is statically generated by defining Alarm information in a CF file.
In the RV850, the kernel initialization module reads the Alarm information definitions from the information file, and initializes these alarms, which are subject to management.

8.3.1 System Services

The system services shown in "14.4.6Alarm management" are used to manipulate alarms dynamically from processing programs.

9. SCHEDULE TABLE MANAGEMENT

This chapter describes the schedule table management functions provided by the RV850.

9.1 Overview

The RV850 provides schedule table management functions as a mechanism to perform processing in synchronization with the change of counter values.

The schedule table management functions perform the expiry action when the count value of a counter associated in Counter identifier "OsScheduleTableCounterRef" reaches the expiry conditions (expiry count value). This expiry action includes activating tasks and setting event masks.

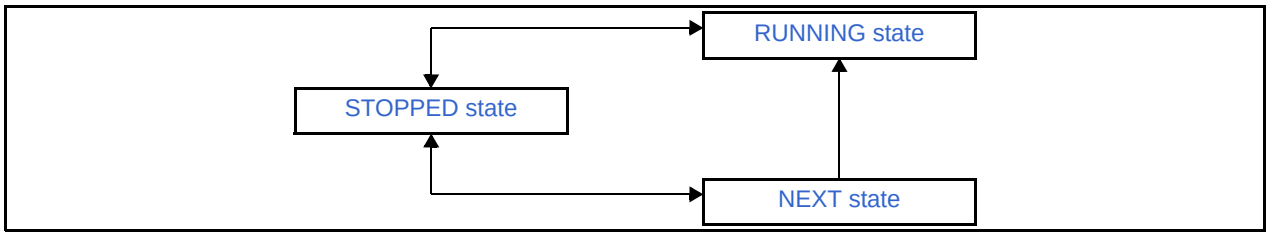
9.2 Schedule Tables

Although only one expiry count value and one expiry action can be defined for an alarm, multiple expiry count values and expiry actions can be defined for a schedule table.

9.2.1 Schedule table states

The RV850 classifies and manages the states that schedule tables can take into five main types.

Figure 9.1 Schedule Table States



(1) STOPPED state
In this state, the schedule table is under the management of the RV850, but it is not subject to schedule counting.

(2) NEXT state
In this state, the schedule table specified in parameter *ScheduleTableID_To* is to be shifted from STOPPED state due to issuance of NextScheduleTable.
A schedule table is shifted from NEXT state to RUNNING state at the timing of when the schedule table (schedule table specified in parameter *ScheduleTableID_From*) currently in RUNNING state has completed schedule counting.

(3) RUNNING state
In this state, the schedule count is running.

9.3 Generation of schedule tables

The RV850 restricts schedule table generation to static generation via the definition of Schedule table information. Consequently, schedule tables cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
A schedule table is statically generated by defining Schedule table information in a CF file.
In the RV850, the kernel initialization module reads the Schedule table information definitions from the information file, and initializes these schedule tables, which are subject to management.

9.4 System Services

The system services shown in "14.4.7Schedule table management" are used to manipulate schedule tables dynamically from processing programs.

10. OS-APPLICATION MANAGEMENT

This chapter describes the OS-Application management functions (only in SC3) provided by the RV850.

10.1 Overview

The RV850 provides OS-Application management functions as a mechanism to group objects (e.g. tasks and counters) used on the system and provide access protection (disables or enables manipulation of objects when system services are issued).

Note that objects are grouped in individual OS-Applications. Access protection is implemented by allowing unconditional access to objects belonging to the same OS-Application and granting individual access to objects belonging to other OS-Applications using OS-Application identifier "OsTaskAccessingApplication", OS-Application identifier "OsCounterAccessingApplication", etc.

10.1.1 Reliability

In the RV850, OS-Applications are divided into the following two types.

(1) Trusted OS-Applications
In the RV850, OS-Applications that are defined as TRUE in Reliability "OsTrusted" are handled as trusted OS-Applications (OS-Applications whose reliability is ensured).
Note that in the RV850, since the reliability of processing programs belonging to a trusted OS-Application is ensured, they are to be operated in supervisor mode (UM bit of PSW is set to 0) with the system protection identifier (SPID bit of MCFG0 register) set to 0, and the memory protection and peripheral I/O protection facilities cannot be applied.

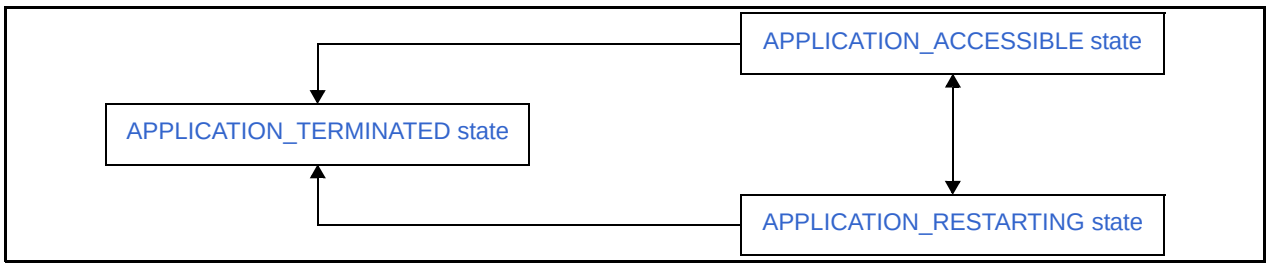
(2) Non-Trusted OS-Applications
In the RV850, OS-Applications that are defined as FALSE in Reliability "OsTrusted" are handled as non-trusted OS-Applications (OS-Applications whose reliability is not ensured).
Note that in the RV850, since the reliability of processing programs belonging to a non-trusted OS-Application is not ensured, they are to be operated in user mode (UM bit of PSW is set to 1) with the system protection identifier (SPID bit of MCFG0 register) specified by SPID "OsApplicationSPID", and the memory protection and peripheral I/O protection functions can be applied.

10.1.2 States

An OS-Application shifts to various states because a system service (e.g. StartOS, TerminateApplication, AllowAccess) was issued from a processing program or because of the value (e.g. APPLICATION_TERMINATED, APPLICATION_RESTARTING) set as the return value of a common hook routine (ProtectionHook).

The RV850 classifies and manages the states that OS-Applications can take into three main types.

Figure 10.1 OS-Application States



(1) APPLICATION_ACCESSIBLE state
After StartOS is issued from the boot process, all OS-Applications defined in CF files are first placed in this state. Even when AllowAccess has been issued from a processing program, the OS-Application to which the processing program that has issued AllowAccess belongs is shifted from APPLICATION_RESTARTING state to this state.

(2) APPLICATION_RESTARTING state
When TerminateApplication (restart option: RESTART) is issued from a processing program, the target OS-Application in APPLICATION_ACCESIBLE state is shifted to this state.

Even when APPLICATION_RESTARTING has been set as the return value of a common hook routine (ProtectionHook), the OS-Application to which the processing program (the processing program in which a protection violation, such as stack overflow, illegal memory access, and exception takes place) calling ProtectionHook belongs is shifted to this state.

(3) APPLICATION_TERMINATED state
When TerminateApplication (restart option: NO_RESTART) is issued from a processing program, the target OS-Application in APPLICATION_ACCESIBLE state is shifted to this state.
Even when APPLICATION_TERMINATED has been set as the return value of a common hook routine (ProtectionHook), the OS-Application to which the processing program (the processing program in which a protection violation, such as stack overflow, illegal memory access, and exception takes place) calling ProtectionHook belongs is shifted to this state.

10.1.3 Memory protection

The RV850 provides a memory protection in order to prevent processing programs (tasks, interrupt service routines, and hook routines) belonging to non-trusted OS-Applications from performing illegal memory access.

If a processing program makes an illegal memory access, this function executes a process (calls the common hook routine ProtectionHook or issues ShutdownOS) according to the definitions of ProtectionHook "OsProtectionHook".

The memory areas that are subject to monitoring by the memory protection of the RV850 are shown below. The memory areas that are subject to monitoring are defined in Memory area identifier "OsSystemMemoryArea", and the type (OS-application specific, common for system) is defined in Memory area identifier "OsAppMemoryAreaNameRef", Memory area identifier "OsMemoryAreaNameRef". A memory area that has not been defined for this function will be handled as a memory area whose access is inhibited.

Table 10.1 Monitored Memory Areas

Memory Area Type		Access Type
Code area	OS-application specific	Read/execution
	Common for system	
Data area	OS-application specific	Read/write
	Common for system	
Stack area	OS-application specific	Read/write
	Extended-task specific	
I/O area		Read/write

Remark 1. For the stack area, no definitions are necessary in Memory area identifier "OsSystemMemoryArea", Memory area identifier "OsAppMemoryAreaNameRef", or Memory area identifier "OsMemoryAreaNameRef".
The stack area used by a non-trusted OS-Application is monitored constantly regardless of the specification of Stack monitoring facilities "OsStackMonitoring"

Remark 2. The threshold (address) of the stack pointer which is to be considered to detect an overflow in the stack area is obtained by adding the maximum value of stack usage by the system services of the RV850 to the top address of the stack area used by tasks or interrupt service routines (category 2).
This is a countermeasure for the memory protection facility not operating while executing a system service of the RV850 because of the transition to supervisor mode even for a non-trusted OS-Application.

10.1.4 Peripheral I/O protection function

The RV850 provides a peripheral I/O protection function that uses the SPID bit (system protection identifier) of the machine configuration register (MCFG0) in order to prevent processing programs (tasks, interrupt service routines, and hook routines) belonging to non-trusted OS-Applications from performing illegal peripheral I/O access.

The RV850 implements the peripheral I/O protection function by rewriting the SPID bit value with the value defined in SPID "OsApplicationSPID" when the OS-Applications are switched.

Remark 1. The I/O areas that are subject to monitoring by the peripheral I/O protection function of the RV850 have to be defined in Memory area identifier "OsAppMemoryAreaNameRef" as memory areas whose access is enabled.

Remark 2. The values to be defined in SPID "OsApplicationSPID" do not have to be on a one-to-one basis with non-trusted OS-Applications.
Consequently, the same SPID "OsApplicationSPID" can be defined for multiple non-trusted OS-Applications.

10.2 Trusted Functions

For trusted OS-Applications, it is possible to assign specific trusted functions to individual OS-Applications.

Trusted functions are called by issuing CallTrustedFunction from the processing program.Since CallTrustedFunction can also be called from other non-trusted OS-Applications, trusted functions can be used when a non-trusted OS-Application is to temporarily perform processing without the protection facilities applied.

The basic form for coding a trusted function in the C language is shown below.

```
TRUSTED ( OsTrustedFunctionName ) {
    ...................
    ...................
}
```

Remark 1. The identifier of the trusted function defined in Identifier "OsTrustedFunctionName" is written in *OsTrustedFunctionName*.

Remark 2. A trusted function can only access objects (e.g. tasks, resources) that can also be accessed from the OS-Application to which the processing program that issued CallTrustedFunction belongs.

Remark 3. Trusted functions operate in supervisor mode; if CallTrustedFunction is issued from a processing program that belongs to a non-trusted OS-Application, the mode switching processing (transition from user mode to supervisor mode) and system protection identifier (SPID bit of MCFG0 register) switching processing (it is set to 0 when a trusted function is being executed) are executed.

Remark 4. Since trusted functions operate in supervisor mode, the memory protection function cannot be applied. Also, since trusted functions operate when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection functions associated with the system protection identifier cannot be applied.

Remark 5. The AUTOSAR specifications have a rule specifying not to perform dispatching to other tasks in the same OS-Application from a trusted function. In the RV850, however, dispatching to other tasks is performed.

Remark 6. The AUTOSAR specifications have a rule disabling acceptance of category 2 interrupts when control is transferred from the processing program that issued CallTrustedFunction to a trusted function. In the RV850, however, no manipulations related with interrupt acceptance are performed.

10.2.1 Processing in trusted functions

When the RV850 transfers control from a processing program to a trusted function, it performs independent pre-processing. It also performs independent post-processing before returning control from a trusted function to a processing program. Consequently, the following points should be noted when coding trusted functions.

(1) Saving/Restoring registers
When the RV850 transfers control to a trusted function, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
The FPSR value is changed to the value defined in FPSR "OsAppDefaultFPSRValue" or FPSR default value "OsDefaultFPSRValue".
Consequently, it is not necessary to code the FPSR save/restore processes when TRUE is defined in FPSR saving/restoring "OsSaveFpuReg".

Remark 1. The FPSR save/restore processes are done only when kernel library libecc2extsc1_fpu.a or libecc2extsc3_fpu.a, which supports the FPU, is linked.

Remark 2. If the trusted function executes floating-point operations using imprecise exception and is needed to complete the operations before tRV850 restores FPSR, it is necessary to issue syncp and synce operation just before the end of the trusted function.

(3) Stack switching
Since the RV850 characterizes a trusted function as an extension of the processing program that issued CallTrustedFunction, stacks are not switched.

Consequently, when estimating the stack size for a processing program that will issue CallTrustedFunction, the size required by the trusted function needs to be considered.

(4) Interrupt acceptance
When the RV850 transfers control to a trusted function, no manipulations related with interrupt acceptance are performed.
Consequently, it is necessary to code the issuance processes, such as EnableAllInterrupts and DisableAllInterrupts, in order to change the status of interrupt acceptance explicitly.

(5) Issuing system services
Since the RV850 characterizes a trusted function as an extension of the processing program that issued CallTrustedFunction, the system services that are allowed to be issued from a trusted function depend on the type of the processing program that issued CallTrustedFunction.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

10.2.2 Registration of trusted functions

The RV850 restricts registration of trusted functions to static registration via the definition of Identifier "OsTrustedFunctionName". Consequently, trusted functions cannot be registered dynamically, for example by issuing system services from a processing program.

(1) Static registration
A trusted function is statically registered by defining Identifier "OsTrustedFunctionName".
In the RV850, the kernel initialization module reads the Identifier "OsTrustedFunctionName" definitions from the information file, and initializes these trusted functions, which are subject to management.

10.2.3 Inherited data of trusted functions

The method of inheriting data between the function calling a trusted function and the trusted function itself is described here.
Assume that trusted function "TFUNC1" is coded in accordance with the basic form as shown below.

```
TRUSTED (TFUNC1) {
    ……
    return;
}
```

In this instance, macro expansion is performed in the format below at compilation.

```
void
TRUSTED_TFUNC1(TrustedFunctionIndexType FunctionIndex,
               TrustedFunctionParameterRefType FunctionParams) {
    ……
    return;
}
```

The first and second parameters specified in the CallTrustedFunction system service can be referenced from trusted function "TFUNC1".

- First parameter FunctionIndex
  This can be referenced in the format of a parameter called FunctionIndex of the TrustedFunctionIndexType type.
- Inherited data pointed to by second parameter FunctionParams
  This can be referenced in the format of referencing the pointer of a parameter called FunctionParams of the TrustedFunctionParameterRefType type.

An example of referencing the inherited data when task "TASK1" calls trusted function "TFUNC1" is shown in the following.

```
/* Structure of inherited data of trusted function TFUNC1 */
struct TFUNC1_parameter_struct {
    int param1;/* Input data */
    int param2;/* Input data */
    int ret_code; /* Input data */
};

/* Allocate inherited data area of trusted function TFUNC1 */
struct TFUNC1_parameter_struct Tfunc1_paramaters;

/* Task */
TASK(TASK1) {
    ……
    ……
     /* Input data to trusted function */
    local_struct.param1 = 1;
    local_struct.param2 = 2;

    /* Calling of trusted function */
    CallTrustedFunction(TFUNC1,\\
                        (TrustedFunctionParameterRefType)&Tfunc1_paramaters);
    /* Acquire output data from trusted function */
    ret_code = Tfunc1_paramaters.value_return;
    ……
    ……
    TerminateTask();
}

/* Trusted function */
TRUSTED (TFUNC1) {
    int p1, p2;

    /* Acquire input data */
    p1 = FunctionParams->param1;
    p2 = FunctionParams->param2;
    ……
    ……
    /* Output data to calling function */
    FunctionParams->value_return = 0x100;

    return;
}
```

10.3 OS-Application-Specific Hook Routines

Three types of hook routines with different usages are supported as OS-Application-specific hook routines in individual OS-Applications.

Remark Non-trusted OS-Application-specific hook routines are to be operated in user mode (UM bit of PSW is set to 1) with the system protection identifier (SPID bit of MCFG0 register) specified by SPID "OsApplication-SPID", and the memory protection and peripheral I/O protection facilities associated with the system protection identifier can be applied.
Trusted OS-Application-specific hook routines are to be operated in supervisor mode (UM bit of PSW is set to 0) with the system protection identifier (SPID bit of MCFG0 register) set to 0, and the memory protection and peripheral I/O protection facilities associated with the system protection identifier cannot be applied.

(1) StartupHook_*OsApplication*
This is a dedicated hook routine for initialization processing that is called when StartOS is issued from a processing program.
The basic form for coding StartupHook_*OsApplication* in the C language is shown below.

```
void
StartupHook_OsApplication ( void ) {
    ..................
    ..................
}
```

Remark 1. The identifier of the OS-Application defined in Identifier "OsApplication" is written in *OsApplication*.

Remark 2. When StartupHook_OsApplication "OsAppStartupHook" is specified as TRUE in multiple sets of OS-Application information, StartupHook_OsApplication is called in the order of appearance in the CF file.

(2) ShutdownHook_*OsApplication*
This is a dedicated hook routine for shutdown processing that is called from a processing program when ShutdownOS is issued.
The basic form for coding ShutdownHook_*OsApplication* in the C language is shown below.

```
void
ShutdownHook_OsApplication ( StatusType Fatalerror ) {
    ..................
    ..................
}
```

Remark 1. The identifier of the OS-Application defined in Identifier "OsApplication" is written in *OsApplication*.

Remark 2. Set parameter *FatalError* to the inherited data specified in parameter *Error* of ShutdownOS.
Note that when the processing program that called this hook routine is the exception/interrupt safety measure process, E_OS_ILLEGAL_EXCEPTION is set to parameter *FatalError*.

Remark 3. When parameter *Fatalerror* is set to E_OS_SYS_ILLEGAL_EXCEPTION, the EIIC or FEIC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcCode in this hook routine, or the EIPC or FEPC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcPC.

Remark 4. When ShutdownHook_OsApplication "OsAppShutdownHook" is specified as TRUE in multiple sets of OS-Application information, ShutdownHook_*OsApplication* is called in the order of appearance in the CF file.

(3) ErrorHook_*OsApplication*
This is a dedicated hook routine for error processing that is called when a system service issued from a processing program terminates abnormally.
The basic form for coding ErrorHook_*OsApplication* in the C language is shown below.

```
void
ErrorHook_OsApplication ( StatusType Error ) {
    ..................
    ..................
}
```

Remark 1. The identifier of the OS-Application defined in Identifier "OsApplication" is written in *OsApplication*.

Remark 2. Set parameter *Error* to the error status of the system service that terminated abnormally.

Remark 3. This hook routine is specific to the OS-Application to which the processing program whose system service terminated abnormally belongs.

Remark 4. When the system service issued from ErrorHook_*OsApplication* terminates abnormally, ErrorHook and ErrorHook_*OsApplication* is not called again.

10.3.1 Processing in OS-Application-specific hook routines

When the RV850 transfers control from a processing program to an OS-Application-specific hook routine, it performs independent pre-processing. It also performs independent post-processing before returning control from an OS-Application-specific hook routine to a processing program. Consequently, the following points should be noted when coding an OS-Application-specific hook routine.

(1) Saving/Restoring registers
When the RV850 transfers control to an OS-Application-specific hook routine, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
When the RV850 transfers control to an OS-Application-specific hook routine, the save/restore processes of the floating-point configuration/status register (FPSR) are not executed.
Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

Remark If the OS-Application-specific hook routine executes floating-point operations using imprecise exception and is needed to complete the operations before transiting into RV850 execution, it is necessary to issue syncp and synce operation just before the end of the OS-Application-specific hook routine.

(3) Stack switching
When the RV850 transfers control to an OS-Application-specific hook routine, it switches to the OS-Application stack defined in OS-Application Stack size "OsAppStackSize".
Consequently, it is not necessary to write code to switch the stack.

(4) Interrupt acceptance
When the RV850 transfers control to an OS-Application-specific hook routine, the acceptance status of category 2 interrupts is changed to disabled (PM*n* bits of PMR are manipulated).

Remark 1. It is prohibited to explicitly manipulate the category 2 interrupt acceptance status from within an OS-Application-specific hook routine.

Remark 2. When the RV850 transfers control to ShutdownHook_*OsApplication*, the ID bit of PSW is manipulated as well as the PM*n* bits of PMR to disable the acceptance of interrupts.

(5) Issuing system services
Only system services that are allowed to be issued from OS-Application-specific hook routines are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

10.3.2 Registration of OS-Application-specific hook routines

The RV850 restricts registration of OS-Application-specific hook routines to static registration via the following definitions. Consequently, OS-Application-specific hook routines cannot be registered dynamically, for example by issuing system services from a processing program.

- StartupHook_OsApplication "OsAppStartupHook"
- ShutdownHook_OsApplication "OsAppShutdownHook"
- ErrorHook_OsApplication "OsAppErrorHook"

(1) Static registration
An OS-Application-specific hook routine is statically registered by defining TRUE in the following definitions.

- StartupHook_OsApplication "OsAppStartupHook"
- ShutdownHook_OsApplication "OsAppShutdownHook"
- ErrorHook_OsApplication "OsAppErrorHook"

In the RV850, the kernel initialization module reads the definitions from the information file, and initializes these OS-Application-specific hook routines, which are subject to management.

10.4 Generation of OS-Applications

The RV850 restricts OS-Application generation to static generation via the definition of OS-Application information. Consequently, OS-Applications cannot be generated dynamically, for example by issuing system services from a processing program.

(1) Static generation
An OS-Application is statically generated by defining OS-Application information in a CF file.
In the RV850, the kernel initialization module reads the OS-Application information definitions from the information file, and initializes these OS-Applications, which are subject to management.

10.5 System Services

The system services shown in "14.4.8OS-Application management" are used to manipulate OS-Applications dynamically from processing programs.

11. OS EXECUTION MANAGEMENT

This chapter describes the OS execution management functions provided by the RV850.

11.1 Overview

The RV850 provides OS execution management functions as a mechanism to perform processing when the RV850 starts up and shuts down.

11.2 Common Hook Routines

In the RV850, six types of hook routines with different usages are supported as common hook routines.

Remark Since common hook routines operate in supervisor mode, the memory protection facility cannot be applied. Also, since common hook routines operate when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection facilities associated with the system protection identifier cannot be applied.

(1) StartupHook
This is a dedicated hook routine for initialization processing that is called when StartOS is issued from a processing program.
The basic form for coding StartupHook in the C language is shown below.

```
void
StartupHook ( void ) {
    ...................
    ...................
}
```

(2) ShutdownHook
This is a dedicated hook routine for shutdown processing that is called when ShutdownOS is issued.
The basic form for coding ShutdownHook in the C language is shown below.

```
void
ShutdownHook ( StatusType Fatalerror ) {
    ...................
    ...................
}
```

Remark 1. Set parameter *FatalError* to the inherited data specified in parameter *Error* of ShutdownOS.
Note that when the processing program that called this hook routine is the exception/interrupt safety measure process, E_OS_ILLEGAL_EXCEPTION is set to parameter *FatalError*.

Remark 2. When parameter *Fatalerror* is set to E_OS_SYS_ILLEGAL_EXCEPTION, the EIIC or FEIC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcCode in this hook routine, or the EIPC or FEPC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcPC.

(3) PostTaskHook
This is a dedicated hook routine for pre-scheduling processing which is called from the scheduler.
The basic form for coding PostTaskHook in the C language is shown below.

```
void
PostTaskHook ( void ) {
    ...................
    ...................
}
```

(4) PreTaskHook
This is a dedicated hook routine for post-scheduling processing which is called from the scheduler.
The basic form for coding PreTaskHook in the C language is shown below.

```
void
PreTaskHook ( void ) {
    ...................
    ...................
}
```

(5) ErrorHook
This is a dedicated hook routine for error processing that is called when a system service issued from a processing program terminates abnormally.
The basic form for coding ErrorHook in the C language is shown below.

```
void
ErrorHook ( StatusType Error ) {
    ..................
    ..................
}
```

Remark 1. Set parameter *Error* to the error status of the system service that terminated abnormally.
See "14.2.2Error status" for details on the error status.

Remark 2. When the system service issued from ErrorHook terminates abnormally, ErrorHook and ErrorHook_*OsApplication* is not called again.

(6) ProtectionHook
This is a dedicated hook routine for the protection-violation handling process called when the RV850 detects a protection violation (e.g. stack overflow, illegal memory access, exception).
The basic form for coding ProtectionHook in the C language is shown below.

```
ProtectionReturnType
ProtectionHook ( StatusType Fatalerror, void *adr ) {
    ..................
    ..................
    return (ProtectionReturnType ReturnType );
}
```

Remark 1. The protection violation type is set in parameter *FatalError*.
The values set in *FatalError* are shown below.

[E_OS_STACKFAULT (0x16)]
A stack overflow was detected.

[E_OS_PROTECTION_MEMORY (0x17)]
An illegal memory access was detected.

Remark 2. In the area specified by parameter *adr*, "0" is stored if parameter *Fatalerror* is E_OS_STACKFAULT or the value of "the status save register when acknowledging FE-level-interrupt (FEPC)" when a protection violation was detected if Fatalerror is E_OS_PROTECTION_MEMORY.

Remark 3. In return value *ReturnType*, specify the processing to be performed by the RV850 after the common hook routine completes its processing.
The values that can be set in *ReturnType* are shown below.

[PRO_TERMINATETASKISR (0x1)]
The process will differ as follows depending on the type of processing program that has generated a protection violation.

[Task]
Shifts the task to SUSPENDED state.
Releases the resources that have been acquired by the task.
Issues EnableAllInterrupts if the task has issued DisableAllInterrupts.
Issues ResumeAllInterrupts if the task has issued SuspendAllInterrupts.
Issues ResumeOSInterrupts if the task has issued SuspendOSInterrupts.
Activates the scheduler.

[Interrupt service routine]
Releases the resources that have been acquired by the task.
Issues EnableAllInterrupts if the task has issued DisableAllInterrupts.
Issues ResumeAllInterrupts if the task has issued SuspendAllInterrupts.
Issues ResumeOSInterrupts if the task has issued SuspendOSInterrupts.
Activates the scheduler.

[Others]
Same processing as PRO_TERMINATEAPPL.

[PRO_TERMINATEAPPL (0x2)]
Executes the process equivalent to TerminateApplication (with the restart option set to NO_RESTART) for the OS-Application to which the processing program that has generated a protection violation belongs.
If no OS-Application is in APPLICATION_ACCESSIBLE state or Task identifier "OsRestartTask" has not been defined, ShutdownOS (with the inherited data set to *Fatalerror*) is issued.

[PRO_SHUTDOWN (0x4)]
ShutdownOS (inherited data: *Fatalerror*) is issued.

[PRO_TERMINATEAPPL_RESTART (0x12)]
Executes the process equivalent to TerminateApplication (with the restart option set to RESTART) for the OS-Application to which the processing program that has generated a protection violation belongs.
If no OS-Application is in APPLICATION_ACCESSIBLE state or Task identifier "OsRestartTask" has not been defined, ShutdownOS (with the inherited data set to *Fatalerror*) is issued.

[Others]
ShutdownOS (inherited data: *Fatalerror*) is issued.

Remark 4. The parameter *adr* is not specified by the AUTOSAR specifications.
This parameter have been uniquely added to the RV850.

11.2.1 Processing in common hook routines

When the RV850 transfers control from a processing program to a common hook routine, it performs independent pre-processing. It also performs independent post-processing before returning control from a common hook routine to a processing program. Consequently, the following points should be noted when coding a common hook routine.

(1) Saving/Restoring registers
When the RV850 transfers control to a common hook routine, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

Remark When the RV850 transfers control to ProtectionHook, the working register for use with FE levels (FEWR) is used without being saved and restored.
Consequently, after control moves to ProtectionHook, the FEWR value becomes undefined.

(2) Saving/Restoring FPSR
When the RV850 transfers control to a common hook routine, the save/restore processes of the floating-point configuration/status register (FPSR) are not executed.
Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

Remark If the common hook routine executes floating-point operations using imprecise exception and is needed to complete the operations before transiting into RV850 execution, it is necessary to issue syncp and synce operation just before the end of the common hook routine.

(3) Stack switching
When the RV850 transfers control to a common hook routine, it switches to the system stack defined in System stack size "OsStackSize".
Consequently, it is not necessary to write code to switch the stack.

Remark If the scalability class is SC3, when the RV850 transfers control to PostTaskHook, PreTaskHook, or ErrorHook, it switches to the OS-Application stack defined in OS-Application Stack size "OsAppStackSize".

(4) Interrupt acceptance
When the RV850 transfers control to a common hook routine, the acceptance status of category 2 interrupts is changed to disabled (PM*n* bits of PMR are manipulated).

Remark 1. It is prohibited to explicitly manipulate the category 2 interrupt acceptance status from within a common hook routine.

Remark 2. When the RV850 transfers control to ShutdownHook and ProtectionHook, in addition to manipulating the PM*n* bits of PMR, the ID bit of PSW is manipulated as the process to disable acceptance of interrupts.

(5) Issuing system services
Only system services that are allowed to be issued from common hook routines are issuable.

Remark See "14.4System Services Reference" for details about the issue scope of each system service.

11.2.2 Registration of common hook routines

The RV850 restricts registration of common hook routines to static registration via the following definitions. Consequently, common hook routines cannot be registered dynamically, for example by issuing system services from a processing program.

- StartupHook "OsStartupHook"
- ShutdownHook "OsShutdownHook"
- PostTaskHook "OsPostTaskHook"
- PreTaskHook "OsPreTaskHook"
- ErrorHook "OsErrorHook"
- ProtectionHook "OsProtectionHook" (only in SC3)

(1) Static registration
A common hook routine is statically registered by defining TRUE in the following definitions.

- StartupHook "OsStartupHook"
- ShutdownHook "OsShutdownHook"
- PostTaskHook "OsPostTaskHook"
- PreTaskHook "OsPreTaskHook"
- ErrorHook "OsErrorHook"
- ProtectionHook "OsProtectionHook" (only in SC3)

In the RV850, the kernel initialization module reads the definitions from the information file, and initializes these common hook routines, which are subject to management.

11.2.3 System Services

The system services shown in "14.4.9OS execution management" are used to manipulate OS execution dynamically from processing programs.

12. SCHEDULE MANAGEMENT

This chapter describes the schedule-management functions provided by the RV850.

12.1 Overview

The RV850 provides schedule-management functions as a mechanism for managing and determining the order of task execution, and assigning processor for the appropriate task to use a device, by directly referencing dynamically changing task status.

The RV850 supports the two types of scheduling methods described below.

(1) Non-preemptive

The scheduler is activated when the task executing processing explicitly discards processor. The start-up conditions of the non-preemptive scheduler are as follows:

- TerminateTask issued
- ChainTask issued
- Schedule issued
- WaitEvent issued
- TerminateApplication issued
- Protection exception (system error exception, memory protection exception, or privilege instruction exception) generated

Remark The scheduler can only be activated by issuing WaitEvent if an event mask that satisfies the request conditions is not set in the target event.

(2) Preemptive

The scheduler is launched in response to some event (trigger). The start-up conditions of the preemptive scheduler are as follows:

- ActivateTask issued
- TerminateTask issued
- ChainTask issued
- Schedule issued
- ReleaseResource issued
- SetEvent issued
- WaitEvent issued
- TerminateApplication issued
- Protection exception (system error exception, memory protection exception, privileged instruction exception) occurred
- An instruction to return from a category 2 interrupt service routine is issued
- Alarm or schedule table expired

12.2 Hook Routines

In the RV850, hook routines are called as pre and post-processing for scheduling processes.

(1) PostTaskHook

This is a dedicated hook routine for pre-scheduling processing which is called from the scheduler.

(2) PreTaskHook

This is a dedicated hook routine for post-scheduling processing which is called from the scheduler.

Remark See "11.2Common Hook Routines" for details about PostTaskHook and PreTaskHook.

12.3 Idle Handler

This is a routine dedicated to idle processing that is extracted for effectively using the low-power support function provided in target devices. It is called from the Scheduler when there are no tasks (READY state or RUNNING state tasks) targeted for RV850 scheduling.

The basic form for coding the idle handler in the C language is shown below.

```
void
IdleHandler ( void ) {
    ...................
    ...................
}
```

Remark 1. Since the idle handler operates in supervisor mode, the memory protection facility cannot be applied. Also, since the idle handler operates when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection facilities associated with the system protection identifier cannot be applied.

Remark 2. A default idle handler is prepared in the RV850.
Consequently, if IdleHandler is not coded, the default idle handler (an empty infinite loop process) will be called.

12.3.1 Processing in idle handler

An independent pre-processing is executed when control is transferred to the idle handler in the RV850. Consequently, the following points should be noted when coding an idle handler.

(1) Saving/Restoring registers
When the RV850 transfers control to an idle handler, the save/restore processes of the work registers are executed according to the C compiler's rules for calling functions.
Consequently, it is not necessary to code the save/restore processes of registers.

(2) Saving/Restoring FPSR
When the RV850 transfers control to an idle handler, the save/restore processes of the floating-point configuration/status register (FPSR) are not executed.
Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly.

(3) Stack switching
When the RV850 transfers control to an idle handler, it switches to the system stack defined in System stack size "OsStackSize".
Consequently, it is not necessary to write code to switch the stack.

(4) Interrupt acceptance
When the RV850 transfers control to an idle handler, the acceptance status of interrupts is changed to enabled (ID bit of PSW is manipulated).

Remark The RV850 prohibits the acceptance status of interrupts from being changed to disabled from an idle handler.

(5) Issuing system services
Issuance of system services is disabled.

13. SYSTEM INITIALIZATION

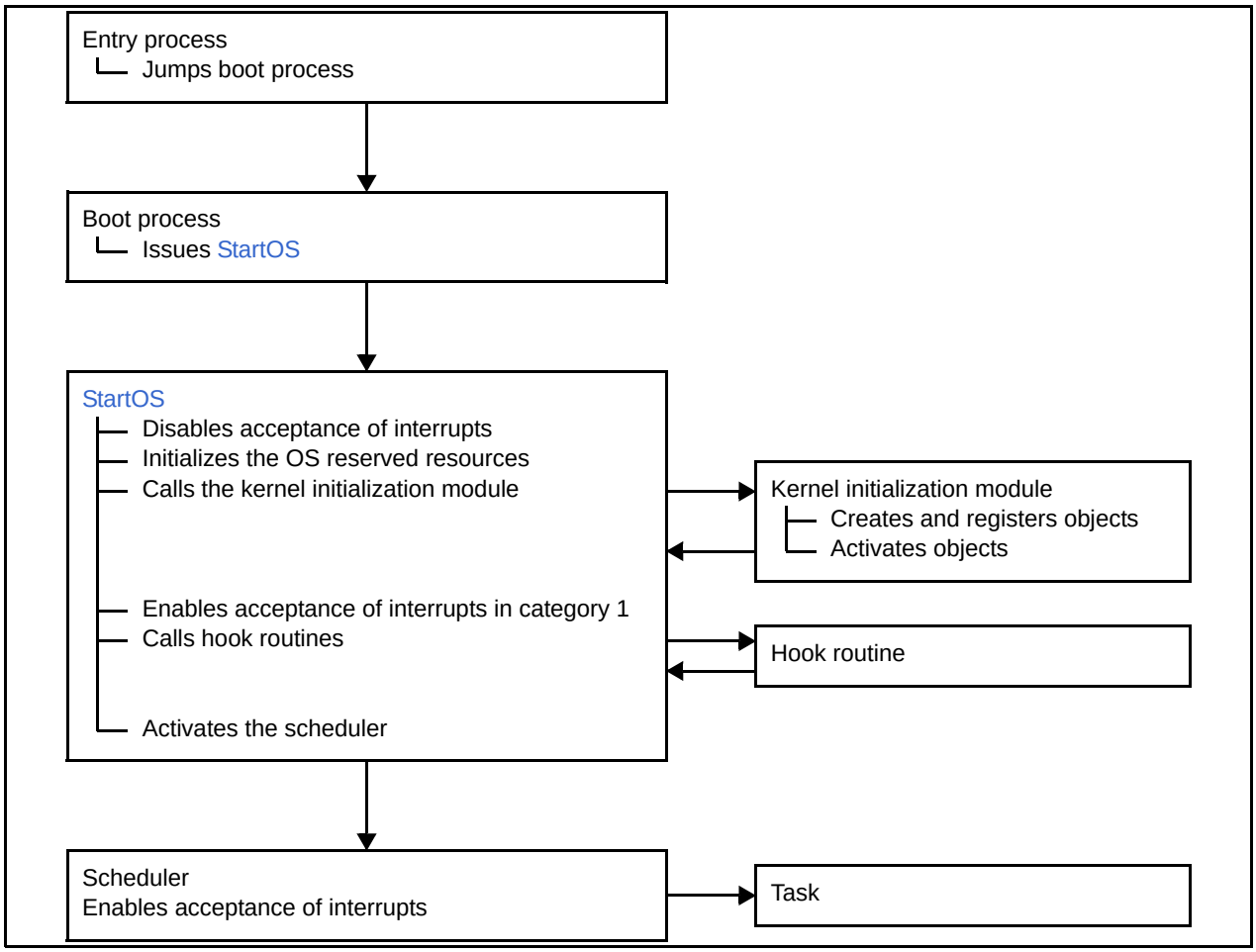
This chapter describes the system initialization process provided by the RV850.

13.1 Overview

The RV850 provides hardware/software initialization processes that are necessary for the RV850 to execute processing. These processes must be executed between the occurrence of a hardware reset and the transfer of control to a processing program (task).

The figure below shows the flow of processes executed from the time of hardware reset occurrence until control is transferred to the processing program (task).

Figure 13.1 System Initialization Processing Sequence



Remark The RV850 does not guarantee correct operation if the user manipulates the EI level interrupt mask register (IMR*m*) to enable the acceptance of the EL-level interrupts defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode" before the hook routines are called through StartOS.

13.2 Entry Process (Direct Branch Method Exception Vector)

The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated.

Remark See "4.5.1Entry process (direct branch method exception vector)" for details about the FE level exception entry process.

13.3 Boot Process

This is a routine dedicated to initialization processing, extracted to initialize the minimum hardware that is required by the RV850 for executing processes. It is called from the branch process (entry process) that is assigned to the address of the handler to which the device forcibly transfers control when a hardware reset is generated.

Remark See "4.2Boot Process" for details about the boot process.

13.4 Kernel Initialization Module

This is a routine dedicated to initialization process extracted for performing the minimum software initialization required by the RV850 to execute processes. It is called within the boot process by issuing StartOS.

The following processes are performed in the kernel initialization module.

(1) Object generation/registration
Generates and registers objects (e.g. tasks, interrupt service routines, and resources), based on information defined in the Task information, Interrupt service routine information, Resource information, and the like.

(2) Object activation
Activates objects (e.g. tasks, alarms, and schedule tables), based on information defined in the Task information, Alarm information, Schedule table information, and the like.

Remark The user does not need to code a kernel initialization module, because the kernel initialization module is included in the functions provided by the RV850.

13.5 Hook Routines

Hook routines are called by the kernel initialization module, and are hook routines dedicated to initialization processing.

Remark 1. When Scalability class "OsScalabilityClass" is scalability class 3 (SC3) in the RV850, common hook routine StartupHook is to be called before OS-Application-specific hook routine StartupHook_*OsApplication* is called.

Remark 2. See "11.2Common Hook Routines" for details about StartupHook, and "10.3OS-Application-Specific Hook Routines" for details about StartupHook_*OsApplication*.

14. SYSTEM SERVICES

This chapter describes the system service functions provided by the RV850.

14.1 Overview

The system services provided by the RV850 are service routines reserved for indirectly operating resources (e.g. management objects and memory area) that are directly managed by the RV850 from processing programs coded by the user.

The following shows the system services provided by the RV850.

(1) Task management
ActivateTask, TerminateTask, ChainTask, Schedule, GetTaskID, GetTaskState

(2) Interrupt handling
EnableAllInterrupts, DisableAllInterrupts, ResumeAllInterrupts, SuspendAllInterrupts, ResumeOSInterrupts, SuspendOSInterrupts

(3) Resource management
GetResource, ReleaseResource

(4) Event management
SetEvent, ClearEvent, GetEvent, WaitEvent

(5) Counter management
IncrementCounter, GetCounterValue, GetElapsedValue

(6) Alarm management
GetAlarmBase, GetAlarm, SetRelAlarm, SetAbsAlarm, CancelAlarm

(7) Schedule table management
StartScheduleTableRel, StartScheduleTableAbs, StopScheduleTable, NextScheduleTable, GetScheduleTableStatus

(8) OS-Application management
GetApplicationID, GetISRID, CallTrustedFunction, CheckISRMemoryAccess, CheckTaskMemoryAccess, CheckObjectAccess, CheckObjectOwnership, TerminateApplication, AllowAccess, GetApplicationState

(9) OS execution management
StartOS, ShutdownOS, GetActiveApplicationMode

(10) Utility functions
InitApplicationInterrupts, _kernel_fv0_InitializeIntService, OSIllegalException_SystemRegister_ExcCode, OSIllegalException_SystemRegister_ExcPC, OSErrorGetServiceId, OSError_SystemService_Parameter

Remark The utility functions InitApplicationInterrupts, _kernel_fv0_InitializeIntService, OSIllegalException_SystemRegister_ExcCode and OSIllegalException_SystemRegister_ExcPC are not specified by the AUTOSAR specifications.
These utility functions have been uniquely added to the RV850.

14.1.1 Calling of system services

The system services provided by the RV850 are implemented as C language functions. Therefore, when a system service is issued from a processing program coded in the C language, the process is performed by calling the system service in a method similar to that for a normal C language function.

Note that when issuing a system service from a processing program coded in the assembly language, the process is performed by setting the parameters and return address in accord with the rules for calling functions of the C compiler package in use immediately before issuing the system service.

If a system service of Interrupt handling is issued before the system initialization process is finished, _kernel_fv0_InitializeIntService needs to be issued before the system service is issued.

Remark 1. The header file definition (include) shown below is necessary in a processing program that issues a system service.

- Os.h: Standard header file

Remark 2. In the RV850, when a system service is issued, a process to switch to the stack (task stack, system stack, etc.) in accord with the processing program type that issued the system service is performed. Accordingly, a process to switch stacks when a system service is issued does not have to be included in the processing program.

Remark 3. If the illegal value is set when SYSCALL instruction is issued, Exception/interrupt safety measure process "_kernel_e_IllegalExcEntry" is called.

14.2 Data Macros

The following shows the data macros used when system services provided by the RV850 are issued.

14.2.1 Data types

Below is described the data type used when system service is issued.
The data types are defined in the header file "Os_types.h" that is called from standard header file "Os.h".

Table 14.1 Data Types

Macro	Type	Description
AccessType	unsigned short	Access privilege
AlarmBaseRefType	AlarmBaseType *	Pointer to the area where the alarm base information is to be stored.
AlarmBaseType	struct _AlarmBaseType	Alarm base information
AlarmType	signed short	Alarm identifier
ApplicationStateRefType	ApplicationStateType *	Pointer to the area where the state of the OS-Application is to be stored.
ApplicationStateType	signed short	State of the OS-Application
ApplicationType	signed short	OS-Application identifier
AppModeType	signed short	Application mode
boolean	unsigned char	Boolean value (TRUE or FALSE)
CounterType	signed short	Counter identifier
EventMaskRefType	EventMaskType *	Pointer to the area where the event mask is to be stored.
EventMaskType	unsigned long	Event mask
float32	float	32-bit floating-point number
float64	double	64-bit floating-point number
ISRType	signed short	Interrupt service routine identifier
MemorySizeType	unsigned long	Size of the memory area
MemoryStartAddressType	unsigned long	Start address of the memory area
ObjectAccessType	signed short	Access privilege
ObjectTypeType	signed short	Object type
OSServiceIdType	signed char	System service identifier
PhysicalTimeType	unsigned long	Time
ProtectionReturnType	signed short	Return value of common hook routine ProtectionHook
RestartType	signed short	Restart option
ResourceType	signed short	Resource identifier
ScheduleTableStatusRefType	ScheduleTableStatusType *	Pointer to the area where the state of the schedule table is to be stored.
ScheduleTableStatusType	signed short	State of the schedule table

Macro	Type	Description
ScheduleTableType	signed short	Schedule table identifier
sint8	signed char	Signed 8-bit integer
sint8_least	signed long	Signed integer (at least 8-bit)
sint16	signed short	Signed 16-bit integer
sint16_least	signed long	Signed integer (at least 16-bit)
sint32	signed long	Signed 32-bit integer
sint32_least	signed long	Signed integer (at least 32-bit)
sint64	signed long long	Signed 64-bit integer
StatusType	unsigned char	Return value of system service
SystemRegisterType	unsigned long	Value of system register
TaskRefType	TaskType *	Pointer to the area where the task identifier is to be stored.
TaskType	signed short	Task identifier
TaskStateRefType	TaskStateType *	Pointer to the area where the state of the task is to be stored.
TaskStateType	signed short	State of the task
TickRefType	TickType *	Pointer to the area where the count value is to be stored or has been stored.
TickType	unsigned long	Count value
TrustedFunctionIndexType	signed short	Trusted function identifier
TrustedFunctionParameterRe-fType	unsigned long *	Pointer to the area where inherited data is stored.
uint8	unsigned char	Unsigned 8-bit integer
uint8_least	unsigned long	Unsigned integer (at least 8-bit)
uint16	unsigned short	Unsigned 16-bit integer
uint16_least	unsigned long	Unsigned integer (at least 16-bit)
uint32	unsigned long	Unsigned 32-bit integer
uint32_least	unsigned long	Unsigned integer (at least 32-bit)
uint64	unsigned long long	Unsigned 64-bit integer

Remark The AUTOSAR specifications have a rule specifying that EventMaskType and TickType are to be 64-bit values. In the RV850, however, EventMaskType and TickType are handled as 32-bit values.

14.2.2 Error status

Below are the macros corresponding to the return values (error status) from system services.
The error status is defined in the header file "Os_error.h" that is called from standard header file "Os.h".

Table 14.2 Error Status

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	The target object cannot be accessed.
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Identifier specification is invalid.
E_OS_LIMIT	0x4	The maximum activation request count has been exceeded.
E_OS_NOFUNC	0x5	The target object cannot be manipulated.
E_OS_RESOURCE	0x6	A disabled operation was performed while acquiring the resource.
E_OS_STATE	0x7	The state of the target object is invalid.
E_OS_VALUE	0x8	Specification of a parameter is invalid.
E_OS_SERVICEID	0x11	Specification of the trusted function identifier is invalid.
E_OS_PARAM_POINTER	0x12	Specification of a parameter is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	Does not have access privilege for the area specified by the parameter.
E_OS_MISSINGEND	0x14	Task ended without issuing TerminateTask or ChainTask.
E_OS_DISABLEDINT	0x15	Issued from a critical section.
E_OS_STACKFAULT	0x16	Overflow of a stack is detected.
E_OS_PROTECTION_MEMORY	0x17	Invalid memory access is detected.
E_OS_SYS_ILLEGAL_EXCEPTION	0x1F	Generation of an undefined interrupt is detected in the Interrupt service routine information.

Remark 1. The AUTOSAR specifications have rules (OS088 and OS093) specifying that error status E_OS_CALLEVEL and E_OS_DISABLEDINT are to be returned when Scalability class "OsScalabilityClass" is scalability class 3 (SC3) or scalability class 4 (SC4). In the RV850, however, this error status is also to be returned when not only the scalability class is SC3 but the scalability class is SC1 with the status type as the extended status.

Remark 2. The error status E_OS_SYS_ILLEGAL_EXCEPTION is not specified by the AUTOSAR specifications. This macro has been uniquely added to the RV850.

14.2.3 Invalid task identifier

Below are the macros corresponding to the numerical values stored in parameter *TaskID* when GetTaskID is issued.
The invalid task identifier is defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.3 Invalid Task Identifier

Macro	Numerical Value	Description
INVALID_TASK	0x7FFF	Target task does not exist.

14.2.4 Task states

Below are the macros corresponding to the numerical values stored in parameter *State* when GetTaskState is issued.
The task states are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.4 Task States

Macro	Numerical Value	Description
SUSPENDED	0x0	SUSPENDED state
READY	0x1	READY state
RUNNING	0x2	RUNNING state
WAITING	0x4	WAITING state

14.2.5 Schedule table states

Below are the macros corresponding to the numerical values stored in parameter *ScheduleStatus* when GetScheduleTableStatus is issued.
The schedule table states are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.5 Schedule Table States

Macro	Numerical Value	Description
SCHEDULETABLE_STOPPED	0x0	STOPPED state
SCHEDULETABLE_NEXT	0x1	NEXT state
SCHEDULETABLE_RUNNING	0x4	RUNNING state

14.2.6 Exit with error (abend)

Below are the macros corresponding to the return values (error codes) from GetApplicationID, GetISRID, and CheckObjectOwnership.
The exit with error is defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.6 Exit with Error (Abend)

Macro	Numerical Value	Description
INVALID_ISR	0x7FFF	Exit with error
INVALID_OSAPPLICATION	0x7FFF	Exit with error

14.2.7 Access privilege types

Below are the macros corresponding to the return values (access privilege types) from CheckISRMemoryAccess and CheckTaskMemoryAccess.

The access privilege types are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.7 Access Privilege Types

Macro	Numerical Value	Description
T_u2_NOACCESS	0x0	No access privileges
T_u2_EXECUTABLE	0x2	Executable
T_u2_READABLE	0x4	Readable
T_u2_WRITEABLE	0x8	Writable
T_u2_STACKSPACE	0x10	Stack area

Remark If there are multiple access privileges, then the return value (access privilege type) will be the logical sum of the values.

14.2.8 Object types

Below are the macros that can be set in parameter *ObjectType* when CheckObjectAccess or CheckObjectOwnership is issued.

The object types are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.8 Object Types

Macro	Numerical Value	Description
OBJECT_TASK	0x1	Task
OBJECT_ISR	0x2	Interrupt service routine
OBJECT_ALARM	0x3	Alarm
OBJECT_RESOURCE	0x4	Resource
OBJECT_COUNTER	0x5	Counter
OBJECT_SCHEDULETABLE	0x6	Schedule table

14.2.9 Checking for access privileges

Below are the macros corresponding to the return values (access privilege check results) from CheckObjectAccess.

The access privilege check results are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.9 Checking for Access Privileges

Macro	Numerical Value	Description
NO_ACCESS	0x0	No access privileges
ACCESS	0x1	Has access privileges.

14.2.10 Restart options

Below are the macros that can be set in the parameter *RestartOption* when TerminateApplication is issued.
The restart options are defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.10 Restart Options

Macro	Numerical Value	Description
NO_RESTART	0x0	Shutdown processing is performed.
RESTART	0x1	The following operations are performed after shutdown processing has been performed. - Shifts the task specified in Task identifier "OsRestartTask" from SUSPENDED state to READY state. - Shifts the target OS-Application from APPLICATION_ACCESSIBLE state to APPLICATION_RESTARTING state.

14.2.11 State of OS-Application

Below are the macros corresponding to the values stored in parameter *Value* when GetApplicationState is issued.
The OS-Application state is defined in the header file "Os_constant.h" that is called from standard header file "Os.h".

Table 14.11 State of OS-Application

Macro	Numerical Value	Description
APPLICATION_ACCESSIBLE	0x0	APPLICATION_ACCESSIBLE state
APPLICATION_RESTARTING	0x1	APPLICATION_RESTARTING state
APPLICATION_TERMINATED	0x2	APPLICATION_TERMINATED state

14.2.12 System service identifiers

Below are the macros corresponding to the return values (system service identifiers) from OSErrorGetServiceId.
The system service identifiers are defined in the header file "Os_service.h" that is called from standard header file "Os.h".

Table 14.12 System Service Identifiers

Macro	Numerical Value	Description
OSServiceID_GetApplicationID	0x0	System service identifier of GetApplicationID
OSServiceID_GetISRID	0x1	System service identifier of GetISRID
OSServiceID_CallTrustedFunction	0x2	System service identifier of CallTrustedFunction
OSServiceId_CheckISRMemoryAccess	0x3	System service identifier of CheckISRMemoryAccess
OSServiceId_CheckTaskMemoryAccess	0x4	System service identifier of CheckTaskMemoryAccess
OSServiceId_CheckObjectAccess	0x5	System service identifier of CheckObjectAccess
OSServiceId_CheckObjectOwnership	0x6	System service identifier of CheckObjectOwnership
OSServiceId_StartScheduleTableRel	0x7	System service identifier of StartScheduleTableRel
OSServiceId_StartScheduleTableAbs	0x8	System service identifier of StartScheduleTableAbs
OSServiceId_StopScheduleTable	0x9	System service identifier of StopScheduleTable
OSServiceId_NextScheduleTable	0xA	System service identifier of NextScheduleTable
OSServiceId_GetScheduleTableStatus	0xE	System service identifier of GetScheduleTableStatus
OSServiceId_IncrementCounter	0xF	System service identifier of IncrementCounter
OSServiceId_GetCounterValue	0x10	System service identifier of GetCounterValue
OSServiceId_GetElapsedValue	0x11	System service identifier of GetElapsedValue
OSServiceId_TerminateApplication	0x12	System service identifier of TerminateApplication
OSServiceID_AllowAccess	0x13	System service identifier of AllowAccess
OSServiceID_GetApplicationState	0x14	System service identifier of GetApplicationState
OSServiceId_StartOS	0x40	System service identifier of StartOS
OSServiceId_ShutdownOS	0x41	System service identifier of ShutdownOS
OSServiceId_GetActiveApplicationMode	0x42	System service identifier of GetActiveApplicationMode
OSServiceId_ActivateTask	0x43	System service identifier of ActivateTask
OSServiceId_TerminateTask	0x44	System service identifier of TerminateTask
OSServiceId_ChainTask	0x45	System service identifier of ChainTask
OSServiceId_Schedule	0x46	System service identifier of Schedule
OSServiceId_GetTaskID	0x47	System service identifier of GetTaskID
OSServiceId_GetTaskState	0x48	System service identifier of GetTaskState

Macro	Numerical Value	Description
OSServiceId_EnableAllInterrupts	0x49	System service identifier of EnableAllInterrupts
OSServiceId_DisableAllInterrupts	0x4A	System service identifier of DisableAllInterrupts
OSServiceId_ResumeAllInterrupts	0x4B	System service identifier of ResumeAllInterrupts
OSServiceId_SuspendAllInterrupts	0x4C	System service identifier of SuspendAllInterrupts
OSServiceId_ResumeOSInterrupts	0x4D	System service identifier of ResumeOSInterrupts
OSServiceId_SuspendOSInterrupts	0x4E	System service identifier of SuspendOSInterrupts
OSServiceId_GetResource	0x4F	System service identifier of GetResource
OSServiceId_ReleaseResource	0x50	System service identifier of ReleaseResource
OSServiceId_SetEvent	0x51	System service identifier of SetEvent
OSServiceId_ClearEvent	0x52	System service identifier of ClearEvent
OSServiceId_GetEvent	0x53	System service identifier of GetEvent
OSServiceId_WaitEvent	0x54	System service identifier of WaitEvent
OSServiceId_GetAlarmBase	0x55	System service identifier of GetAlarmBase
OSServiceId_GetAlarm	0x56	System service identifier of GetAlarm
OSServiceId_SetRelAlarm	0x57	System service identifier of SetRelAlarm
OSServiceId_SetAbsAlarm	0x58	System service identifier of SetAbsAlarm
OSServiceId_CancelAlarm	0x59	System service identifier of CancelAlarm

14.2.13 Counter information

Below are the macros corresponding to the Counter information.

The counter information is defined in the information file (kernel macro file) that is generated when the configurator executes CF files.

Table 14.13 Counter Information

Macro	Description
OSMAXALLOWEDVALUE_*OsCounter*	Maximum count value "OsCounterMaxAllowedValue" (unit: tick)
OSMINCYCLE_*OsCounter*	Minimum cycle value "OsCounterMinCycle" (unit: tick)
OSTICKSPERBASE_*OsCounter*	Basic count value "OsCounterTicksPerBase" (unit: tick)
OSMAXALLOWEDVALUE	Maximum count value "OsCounterMaxAllowedValue" (unit: tick) of system counter (when SYS_COUNTER is specified for Identifier "OsCounter")
OSMINCYCLE	Minimum cycle value "OsCounterMinCycle" (unit: tick) of system counter (when SYS_COUNTER is specified for Identifier "OsCounter")
OSTICKSPERBASE	Basic count value "OsCounterTicksPerBase" (unit: tick) of system counter (when SYS_COUNTER is specified for Identifier "OsCounter")
OSTICKDURATION	Basic time (unit: ns) of system counter (when SYS_COUNTER is specified for Identifier "OsCounter") Calculation result of <10^9 * Number of seconds per tick "OsSecondsPerTick" * Basic count value "OsCounterTicksPerBase">
OS_TICK2NS_*OsCounter* (*Value*)	Converts parameter *Value* (unit: tick) into a value in nanoseconds. Calculation result of <*Value* * 10^9 * Number of seconds per tick "OsSecondsPerTick">
OS_TICK2US_*OsCounter* (*Value*)	Converts parameter *Value* (unit: tick) into a value in microseconds. Calculation result of <*Value* * 10^6 * Number of seconds per tick "OsSecondsPerTick">
OS_TICK2MS_*OsCounter* (*Value*)	Converts parameter *Value* (unit: tick) into a value in milliseconds. Calculation result of <*Value* * 10^3 * Number of seconds per tick "OsSecondsPerTick">
OS_TICK2SEC_*OsCounter* (*Value*)	Converts parameter *Value* (unit: tick) into a value in seconds. Calculation result of <*Value* * Number of seconds per tick "OsSecondsPerTick">

14.2.14 Checking for access privileges

Below are the macros corresponding to the access privileges.
The access privileges are defined in the header file "Os_constant.h" that is called from the standard header file "Os.h".

Table 14.14 Checking for Access Privileges

Macro	Description
OSMEMORY_IS_EXECUTEABLE (AccessType *Type*)	Checks whether the memory area is executable with parameter *Type* set to the value returned from CheckISRMemoryAccess or CheckTaskMemoryAccess.
OSMEMORY_IS_READABLE (AccessType *Type*)	Checks whether the memory area is readable with parameter *Type* set to the value returned from CheckISRMemoryAccess or CheckTaskMemoryAccess.
OSMEMORY_IS_WRITEABLE (AccessType *Type*)	Checks whether the memory area is writable with parameter *Type* set to the value returned from CheckISRMemoryAccess or CheckTaskMemoryAccess.
OSMEMORY_IS_STACKSPACE (AccessType *Type*)	Checks whether the memory area is a stack area with parameter *Type* set to the value returned from CheckISRMemoryAccess or CheckTaskMemoryAccess.

14.3 Data Structures

The data structure used when a system service provided from the processing program by the RV850 is issued is described below.

14.3.1 Alarm base information

Below is the alarm base information stored in parameter *Info* when GetAlarmBase is issued.
The alarm base information is defined in the header file "Os_types.h" that is called from standard header file "Os.h".

Figure 14.1 Alarm Base Information

```
struct _AlarmBaseType {
    TickType maxallowedvalue; /* Maximum count value "OsCounterMaxAllowedValue" */
    TickType mincycle;        /* Minimum cycle value "OsCounterMinCycle" */
    TickType ticksperbase;    /* Basic count value "OsCounterTicksPerBase" */
};

typedef struct _AlarmBaseType AlarmBaseType;
typedef AlarmBaseType *AlarmBaseRefType;
```

(1) maxallowedvalue
This stores the Maximum count value "OsCounterMaxAllowedValue" of the counter associated with the alarm specified in the GetAlarmBase parameter *AlarmID*.

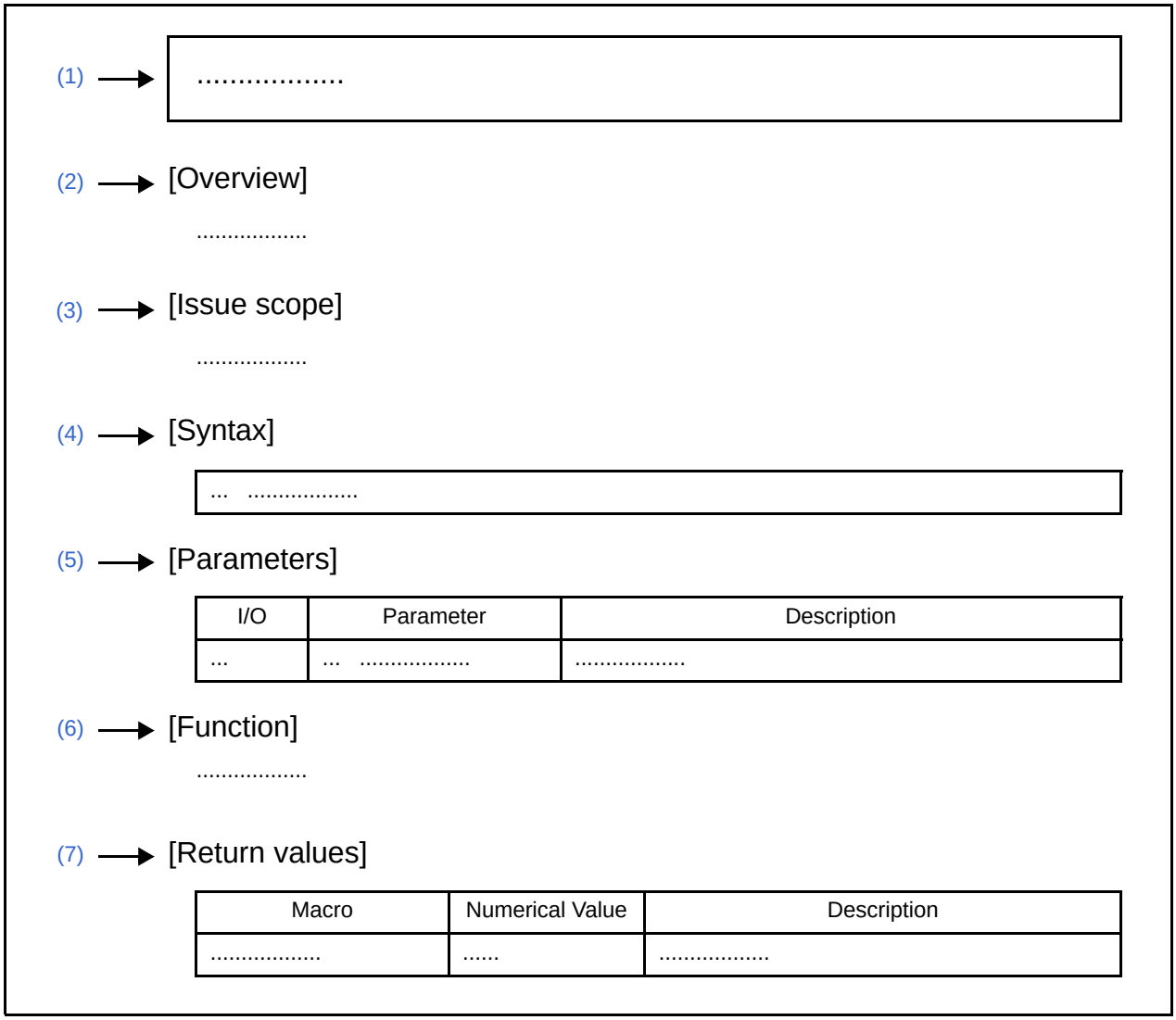
(2) mincycle
This stores the Minimum cycle value "OsCounterMinCycle" of the counter associated with the alarm specified in the GetAlarmBase parameter *AlarmID*.

(3) ticksperbase
This stores the Basic count value "OsCounterTicksPerBase" of the counter associated with the alarm specified in the GetAlarmBase parameter *AlarmID*.

14.4 System Services Reference

This describes the system services provided by the RV850, using the following format.

Figure 14.2 Description Format of System Services



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

(2) [Overview]
Provides an overview of the system service.

(3) [Issue scope]
This shows the types of processing programs that can issue the system service, and where they can do so.

(a) Boot process
This is a routine dedicated to initialization processing, extracted to initialize the minimum hardware that is required by the RV850 for executing processes.

(b) Tasks
Tasks are processing routines that are not executed unless the state is manipulated using a system service, or if conditions defined in a CF file are met.
If the ability to issue the system service depends on the type of the task, then the tasks able to issue it are shown in the following notation.

- If the system service can only be issued from an extended task
  Task (extended)

(c) Interrupt service routines
Interrupt service routines are dedicated to interrupt processes that are called when an interrupt is generated.
If the ability to issue the system service depends on the category of the interrupt service routine, then the interrupt service routines that can issue it are shown in the following notation.

- If the system service can only be issued from category 2 interrupt service routines
  Interrupt service routine (category 2)

(d) Alarm callback
This is a routine dedicated to the expiry action, called when the alarm has expired.

(e) Common hook routines
Common hook routines for all OS-Applications.
If the ability to issue the system service depends on the type of the common hook routine (StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook), then the common hook routines able to issue it are shown in the following notation.

- If the system service can only be issued from PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook
  Common hook routine (PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook)

(f) OS-Application-specific hook routine
Hook routines specific to each OS-Application.
If the ability to issue the system service depends on the type of the OS-Application-specific hook routine (StartupHook_*OsApplication*, ShutdownHook_*OsApplication*, ErrorHook_*OsApplication*), then the OS-Application-specific hook routines able to issue it are shown in the following notation.

- If the system service can only be issued from StartupHook_*OsApplication*, ErrorHook_*OsApplication*
  OS-Application-specific hook routine (StartupHook_*OsApplication*, ErrorHook_*OsApplication*)

(g) Critical sections
Scope for which acceptance of the following interrupts is disabled

- From issuance of DisableAllInterrupts until issuance of EnableAllInterrupts
- From issuance of SuspendAllInterrupts until issuance of ResumeAllInterrupts
- From issuance of SuspendOSInterrupts until issuance of ResumeOSInterrupts

(4) [Syntax]
Gives the specification format when the system service is issued from a processing program coded in the C language.

(5) [Parameters]
The parameters of the system service are shown in the following format.

I/O	Parameter	Description
(a)	(b)	(c)

(a) I/O column
Parameter type

- I ... Input parameter to RV850
- O ... Output parameter from RV850

(b) Parameter column
Data type of the parameter

(c) Description column
Description of the parameter

(6) [Function]
Provides an overview of the function of the system service.

(7) [Return values]
The return value from the system service is shown in the following format.

Macro	Numerical Value	Description
(a)	(b)	(c)

(a) Macro column
Macro return value

(b) Numerical Value column
Numeric return value

(c) Description column
Description of the return value

14.4.1 Task management

The following shows the system services for task management provided by the RV850.

Table 14.15 System Services for Task Management

Name of System Service	Function Overview
ActivateTask	Activates the task.
TerminateTask	Terminates the task.
ChainTask	Terminates and activates tasks.
Schedule	Activates the scheduler.
GetTaskID	Gets the task identifier of the task that has been shifted to RUNNING state.
GetTaskState	Gets the current state of the task.

ActivateTask

[Overview]

Activates the task.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType ActivateTask ( TaskType TaskID );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier

[Function]

Shifts the target task (the task specified in parameter *TaskID*) from SUSPENDED state to READY state.
As the target task is shifted from SUSPENDED state to READY state, the target task is queued at the end of the ready queue corresponding to the priority.

Remark 1. If this system service is issued from a task whose Scheduling attribute "OsTaskSchedule" is a preemptive property (FULL), the scheduler is activated when the state of the task has been manipulated (the target task is shifted from SUSPENDED state to READY state).

Remark 2. When the type of the target task is basic task, the state of the task will be manipulated (the target task is shifted from SUSPENDED state to READY state) and the activation request counter will be incremented (0x1 will be added to the activation request counter).
When the target task is shifted to a state other than SUSPENDED state (i.e., READY state or RUNNING state), then the state of the task will not be manipulated, the task will be queued at the end of the ready queue corresponding to the priority, and the activation request counter will be incremented.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target task. (Only in SC3) - The OS-Application to which the target task belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *TaskID* is invalid.

Macro	Numerical Value	Description
E_OS_LIMIT	0x4	Exit with error - The target task is an extended task in READY state, RUNNING state, or WAITING state. - The activation request count has exceeded Maximum activation request count "OsTaskActivation" of the target task.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

TerminateTask

[Overview]

Terminates the task.

[Issue scope]

Tasks

[Syntax]

```
StatusType TerminateTask ( void );
```

[Parameters]

None

[Function]

Shifts the current task (the task that issued this system service) from RUNNING state to SUSPENDED state.
As the current task is shifted from RUNNING state to SUSPENDED state, the current task is unlinked from the ready queue corresponding to the priority. The scheduler is activated when the state of the task has been manipulated (the current task is shifted from RUNNING state to SUSPENDED state).

Remark 1. When the type of the current task is basic task, the state of the task will be manipulated (the current task is shifted from RUNNING state to SUSPENDED state) and the activation request counter will be decremented (0x1 is subtracted from the activation request counter).
When the subtraction result of the activation request counter is other than 0x0, the state of the task will be manipulated (the current task is shifted from SUSPENDED state to READY state) and the current task will be shifted to the READY state again. The scheduler is activated when the state of the task has been manipulated (the current task is shifted from SUSPENDED state to READY state).

Remark 2. If the current task had acquired any internal resources, the state of the task will be manipulated (the current task is shifted from RUNNING state to SUSPENDED state), internal resources will be released, and the current priority will be changed (current priority of current task is returned to Initial priority "OsTaskPriority").
When the ceiling value of the internal resource is INTPRI*x*, the process to enable the acceptance of interrupts will also be performed (PM*n* bits of PMR are manipulated).

Remark 3. When a task for which scalability class 3 (SC3) is defined for Scalability class "OsScalabilityClass" is terminated without this system service or ChainTask being issued, if a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the task, a hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

[Return values]

Macro	Numerical Value	Description
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_RESOURCE	0x6	Current task has acquired normal resources.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

ChainTask

[Overview]

Terminates and activates tasks.

[Issue scope]

Tasks

[Syntax]

```
StatusType ChainTask ( TaskType TaskID );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier

[Function]

Shifts the current task (the task that issued this system service) from RUNNING state to SUSPENDED state and shifts the target task (the task specified in parameter *TaskID*) from SUSPENDED state to READY state.

As the current task is shifted from RUNNING state to SUSPENDED state, the current task is unlinked from the ready queue corresponding to the priority. As the target task is shifted from SUSPENDED state to READY state, the target task is queued at the end of the ready queue corresponding to the priority.

The scheduler is activated when the state of the tasks has been manipulated (the current task is shifted from RUNNING state to SUSPENDED state and target task is shifted from SUSPENDED state to READY state).

Remark 1. When the current task is specified in parameter *TaskID*, the current task is shifted from RUNNING state to READY state without being shifted to SUSPENDED state.
The scheduler is activated when the state of the task has been manipulated (the current task is shifted from RUNNING state to READY state).

Remark 2. When the type of the current task is basic task, the state of the task will be manipulated (the current task is shifted from RUNNING state to SUSPENDED state) and the activation request counter will be decremented (0x1 is subtracted from the activation request counter).
When the subtraction result of the activation request counter is other than 0x0, the state of the task will be manipulated (the current task is shifted from SUSPENDED state to READY state) and the current task will be shifted to the READY state again.
The scheduler is activated when the state of the task has been manipulated (the current task is shifted from SUSPENDED state to READY state).

Remark 3. When the type of the target task is basic task, the state of the task will be manipulated (the target task is shifted from SUSPENDED state to READY state) and the activation request counter will be incremented (0x1 is added to the counter).
When the target task has already been shifted from SUSPENDED state to READY state or RUNNING state, the state of the task will not be manipulated, the task will be queued at the end of the ready queue corresponding to the priority, and the activation request counter will be incremented.

Remark 4. If the current task had acquired any internal resources, the state of the task will be manipulated (the current task is shifted from RUNNING state to SUSPENDED state or current task is shifted from RUNNING state to READY state), internal resources will be released, and the current priority will be changed (the current priority of current task is returned to Initial priority "OsTaskPriority").
When the ceiling value of the internal resource is INTPRI*x*, the process to enable the acceptance of interrupts will also be performed (PM*n* bits of PMR are manipulated).

Remark 5. When a task for which scalability class 3 (SC3) is defined for Scalability class "OsScalabilityClass" is terminated without TerminateTask or this system service being issued, if a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the task, a hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

[Return values]

Macro	Numerical Value	Description
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the current task belongs does not have access privileges for the target task. (Only in SC3) - The OS-Application to which the target task belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *TaskID* is invalid.
E_OS_LIMIT	0x4	Exit with error - The target task is an extended task in READY state, RUNNING state, or WAITING state. - The activation request count has exceeded Maximum activation request count "OsTaskActivation" of the target task.
E_OS_RESOURCE	0x6	Current task has acquired normal resources.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

Schedule

[Overview]

Activates the scheduler.

[Issue scope]

Tasks

[Syntax]

```
StatusType Schedule ( void );
```

[Parameters]

None

[Function]

Activates the scheduler.

Remark If the current task had acquired any internal resources, internal resources will be released and the current priority will be changed (current priority of current task is returned to the priority that was set before acquiring the resource). After this, the scheduler will be activated.
When the ceiling value of the internal resource is INTPRI*x*, the process to enable the acceptance of interrupts will also be performed (PM*n* bits of PMR are manipulated).

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_RESOURCE	0x6	Current task has acquired normal resources.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetTaskID

[Overview]

Gets the task identifier of the task that has been shifted to RUNNING state.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetTaskID ( TaskRefType TaskID );
```

[Parameters]

I/O	Parameter	Description
O	`TaskRefType` *`TaskID;`*	Pointer to the area where the acquired task identifier is to be stored

[Function]

Gets the task identifier of the target task (the task that changes to RUNNING state when this system service is issued) and stores it in the area specified in parameter *TaskID*.

Remark 1. If there is no target task when this system service is issued, INVALID_TASK (0x7FFF) is stored in the area specified in parameter *TaskID*.

Remark 2. The correspondence between the number acquired by issuing this system service and Identifier "OsTask" defined in the Task information during configuration is defined in the SIT file.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_PARAM_POINTER	0x12	Specification of parameter *TaskID* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *TaskID*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetTaskState

[Overview]

Gets the current state of the task.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (PostTaskHook, PreTaskHook, ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetTaskState ( TaskType TaskID, TaskStateRefType State );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier
O	`TaskStateRefType` *State*`;`	Pointer to the area where the acquired current state is to be stored

[Function]

Gets the current state of the target task (the task specified in parameter *TaskID*) and stores it in the area specified in parameter *State*.

The value stored in parameter *State* depends on the type of the current state, as shown below.

Macro	Numerical Value	Description
SUSPENDED	0x0	SUSPENDED state
READY	0x1	READY state
RUNNING	0x2	RUNNING state
WAITING	0x4	WAITING state

Remark The current state stored in the area specified in parameter *State* is not affected by the value held in the activation request counter of the target task.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal Termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target task. (Only in SC3) - The OS-Application to which the target task belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)

Macro	Numerical Value	Description
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *TaskID* is invalid.
E_OS_PARAM_POINTER	0x12	Specification of parameter *State* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *State*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.2 Interrupt handling

The following shows the system services for interrupt handling provided by the RV850.

Table 14.16 System Services for Interrupt Handling

Name of System Service	Function Overview
EnableAllInterrupts	Enables the acceptance of interrupts (without nesting management).
DisableAllInterrupts	Disables the acceptance of interrupts (without nesting management).
ResumeAllInterrupts	Enables the acceptance of interrupts (with nesting management).
SuspendAllInterrupts	Disables the acceptance of interrupts (with nesting management).
ResumeOSInterrupts	Enables the acceptance of category 2 interrupts (with nesting management).
SuspendOSInterrupts	Disables the acceptance of category 2 interrupts (with nesting management).

EnableAllInterrupts

[Overview]

Enables the acceptance of interrupts (without nesting management).

[Issue scope]

Boot process, tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void EnableAllInterrupts ( void );
```

[Parameters]

None

[Function]

This allows the acceptance of interrupts that were disabled by issuing DisableAllInterrupts.

Remark 1. This system service manipulates the ID bit of the program status word (PSW), as a process to enable the acceptance of interrupts.

Remark 2. Issuing this system service terminates the critical section that was started by issuing DisableAllInterrupts.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued between issuance of SuspendAllInterrupts and issuance of ResumeAllInterrupts, it will not be handled as an error and no processing will be performed.

Remark 5. If this system service is issued during the boot process (before issuing StartOS) or after issuing ShutdownOS, the process to enable the acceptance of interrupts will be forcibly performed (ID bit of PSW is manipulated).
To issue this system service during the boot process (before issuing StartOS), _kernel_fv0_InitializeIntService needs to be issued before this system service is issued.

Remark 6. The AUTOSAR specifications have a rule (OS092) specifying that if DisableAllInterrupts is not issued before issuance of this system service when Scalability class "OsScalabilityClass" is scalability class 3 (SC3) or scalability class 4 (SC4), it will not be handled as an error and no processing will be performed. In the RV850, however, if DisableAllInterrupts is not issued before issuance of this system service, regardless of the definitions in Scalability class "OsScalabilityClass", it will not be handled as an error and no processing will be performed.

[Return values]

None

DisableAllInterrupts

[Overview]

Disables the acceptance of interrupts (without nesting management).

[Issue scope]

Boot process, tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void DisableAllInterrupts ( void );
```

[Parameters]

None

[Function]

Disables the acceptance of interrupts.

Remark 1. This system service manipulates the ID bit of the program status word (PSW), as a process to disable the acceptance of interrupts.

Remark 2. Issuing this system service starts a critical section.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued between issuance of SuspendAllInterrupts and issuance of ResumeAllInterrupts, it will not be handled as an error and no processing will be performed.

Remark 5. It is assumed that this system service will be used with EnableAllInterrupts.
After this system service has been issued, EnableAllInterrupts should be issued in the same processing program.
If this system service is re-issued before issuance of EnableAllInterrupts, it will not be handled as an error and no processing will be performed.

Remark 6. The operation when this system service was issued from a processing program but the processing program terminated before EnableAllInterrupts could be issued will be different according to the definitions in Scalability class "OsScalabilityClass", as follows:

- Scalability class 1 (SC1)
  Correct operation cannot be guaranteed.
- Scalability class 3 (SC3)
  The process will differ as follows depending on the type of processing program.

  [Task]
  Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section.
  If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

  [Interrupt service routine (category 2)]
  Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section.

If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter.

[OS-Application-specific hook routine StartupHook_*OsApplication*]
Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section.

[OS-Application-specific hook routine ShutdownHook_*OsApplication*]
Terminates the critical section.

[Others]
Correct operation is not guaranteed.

The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.

Remark 7. If this system service is issued during the boot process (before issuing StartOS) or after issuing ShutdownOS, the process to enable the acceptance of interrupts will be forcibly performed (ID bit of PSW is manipulated).
To issue this system service during the boot process (before issuing StartOS), _kernel_fv0_InitializeIntService needs to be issued before this system service is issued.

[Return values]

None

ResumeAllInterrupts

[Overview]

Enables the acceptance of interrupts (with nesting management).

[Issue scope]

Boot process, tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void ResumeAllInterrupts ( void );
```

[Parameters]

None

[Function]

Subtracts 0x1 from the disable request counter (dedicated to this system service and SuspendAllInterrupts).
When the subtraction result of the disable request counter is 0x0, the disable request counter will be decremented (0x1 is subtracted from the disable request counter) and the acceptance of interrupts that was disabled by issuance of SuspendAllInterrupts will be enabled.

Remark 1. This system service manipulates the ID bit of the program status word (PSW), as a process to enable the acceptance of interrupts.

Remark 2. When the subtraction result of the disable request counter is 0x0, the critical section that was started by issuing SuspendAllInterrupts is terminated.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued between issuance of DisableAllInterrupts and issuance of EnableAllInterrupts, it will not be handled as an error and no processing will be performed.

Remark 5. If this system service is issued during the boot process (before issuing StartOS) or after issuing ShutdownOS, only the process to enable the acceptance of interrupts will be performed (ID bit of PSW is manipulated) and the disable request counter will not be decremented (0x1 is not subtracted from the disable request counter).
To issue this system service during the boot process (before issuing StartOS), _kernel_fv0_InitializeIntService needs to be issued before this system service is issued.

Remark 6. The AUTOSAR specifications have a rule (OS092) specifying that if SuspendAllInterrupts is not issued before issuance of this system service when Scalability class "OsScalabilityClass" is scalability class 3 (SC3) or scalability class 4 (SC4), it will not be handled as an error and no processing will be performed. In the RV850, however, if SuspendAllInterrupts is not issued before issuance of this system service, regardless of the definitions in Scalability class "OsScalabilityClass", it will not be handled as an error and no processing will be performed.

[Return values]

None

SuspendAllInterrupts

[Overview]

Disables the acceptance of interrupts (with nesting management).

[Issue scope]

Boot process, tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void SuspendAllInterrupts ( void );
```

[Parameters]

None

[Function]

Adds 0x1 to the disable request counter (dedicated to this system service and ResumeAllInterrupts, maximum disable request count: 127).

If the disable request counter is 0x0 when this system service is issued, the disable request counter will be incremented (0x1 is added to the disable request counter) and the acceptance of interrupts will be disabled.

Remark 1. This system service manipulates the ID bit of the program status word (PSW), as a process to disable the acceptance of interrupts.

Remark 2. When the disable request counter is 0x0 when this system service is issued, issuance of this system service starts a critical section.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued between issuance of DisableAllInterrupts and issuance of EnableAllInterrupts, it will not be handled as an error and no processing will be performed.

Remark 5. It is assumed that this system service will be used with ResumeAllInterrupts.
After this system service has been issued, ResumeAllInterrupts should be issued in the same processing program.
If this system service is re-issued before issuance of ResumeAllInterrupts, only the disable request counter will be incremented (0x1 is added to the disable request counter) and the process to disable the acceptance of interrupts will not be performed (ID bit of PSW is not manipulated).
If the re-issuance of this system service causes the disable request counter (maximum disable request count: 127) to overflow, it will not be handled as an error and no processing will be performed.

Remark 6. The operation when this system service was issued from a processing program but the processing program terminated before ResumeAllInterrupts could be issued will be different according to the definitions in Scalability class "OsScalabilityClass", as follows:

- Scalability class 1 (SC1)
  Correct operation cannot be guaranteed.
- Scalability class 3 (SC3)
  The process will differ as follows depending on the type of processing program.

  [Task]
  Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.

If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

[Interrupt service routine (category 2)]
Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.
If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter.

[OS-Application-specific hook routine StartupHook_*OsApplication*]
Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.

[OS-Application-specific hook routine ShutdownHook_*OsApplication*]
Terminates the critical section.

[Others]
Correct operation is not guaranteed.

The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.

Remark 7. If this system service is issued during the boot process (before issuing StartOS) or after issuing ShutdownOS, only the process to disable the acceptance of interrupts will be performed (ID bit of PSW is manipulated) and the disable request counter will not be incremented (0x1 is not added to the disable request counter).
To issue this system service during the boot process (before issuing StartOS), _kernel_fv0_InitializeIntService needs to be issued before this system service is issued.

[Return values]

None

ResumeOSInterrupts

[Overview]

Enables the acceptance of category 2 interrupts (with nesting management).

[Issue scope]

Tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void ResumeOSInterrupts ( void );
```

[Parameters]

None

[Function]

Subtracts 0x1 from the disable request counter (dedicated to this system service and SuspendOSInterrupts).
When the subtraction result of the disable request counter is 0x0, the disable request counter will be decremented (0x1 is subtracted from the disable request counter) and the acceptance of interrupts that was disabled by issuance of SuspendOSInterrupts will be enabled.

Remark 1. This system service manipulates the PM*n* bits of the priority mask register (PMR), as a process to enable the acceptance of interrupts.
The PM*n* bits to be manipulated are bits corresponding to the following priorities and bits corresponding to priorities lower than the following priorities.

- Initial priority "OsIsrPriority" which is defined to be used for an interrupt service routine (category 2)
- Priority "OsCounterPriority" which is defined to be used for a hardware counter

Remark 2. When the subtraction result of the disable request counter is 0x0, the critical section that was started by issuing SuspendOSInterrupts is terminated.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued from an alarm callback, common hook routine (StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, or ErrorHook), or OS-Application-specific hook routine, only the disable request counter will be decremented (0x1 is subtracted from the disable request counter) and the acceptance of interrupts will not be enabled (PM*n* bits of PMR are not manipulated).

Remark 5. If this system service is issued when GetResource (with the ceiling value set to INTPRI*x*) has been issued before SuspendOSInterrupts is issued, the priority mask register (PMR) is set to the value before SuspendOSInterrupts is issued (the value indicating that the acceptance of the interrupt sources corresponding to INTPRI0 to INTPRI*x* is disabled).

Remark 6. The AUTOSAR specifications have a rule (OS092) specifying that if SuspendOSInterrupts is not issued before issuance of this system service when Scalability class "OsScalabilityClass" is scalability class 3 (SC3) or scalability class 4 (SC4), it will not be handled as an error and no processing will be performed. In the RV850, however, if SuspendOSInterrupts is not issued before issuance of this system service, regardless of the definitions in Scalability class "OsScalabilityClass", it will not be handled as an error and no processing will be performed.

[Return values]

None

SuspendOSInterrupts

[Overview]

Disables the acceptance of category 2 interrupts (with nesting management).

[Issue scope]

Tasks, interrupt service routines, alarm callbacks, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
void SuspendOSInterrupts ( void );
```

[Parameters]

None

[Function]

Adds 0x1 to the disable request counter (dedicated to this system service and ResumeOSInterrupts, maximum disable request count: 127).

If the disable request counter is 0x0 when this system service is issued, the disable request counter will be incremented (0x1 is added to the disable request counter) and the acceptance of interrupts will be disabled.

Remark 1. This system service manipulates the PM*n* bits of the priority mask register (PMR), as a process to disable the acceptance of interrupts.
The PM*n* bits to be manipulated are bits corresponding to the following priorities and bits corresponding to priorities lower than the following priorities.

- Initial priority "OsIsrPriority" which is defined to be used for an interrupt service routine (category 2)
- Priority "OsCounterPriority" which is defined to be used for a hardware counter

Remark 2. When the disable request counter is 0x0 when this system service is issued, issuance of this system service starts a critical section.

Remark 3. If this system service is issued from the common hook routine (ProtectionHook), it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued from an alarm callback, common hook routine (StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, or ErrorHook), or OS-Application-specific hook routine, only the disable request counter will be incremented (0x1 is added to the disable request counter) and the acceptance of interrupts will not be disabled (PM*n* bits of PMR are not manipulated).

Remark 5. It is assumed that this system service will be used with ResumeOSInterrupts.
After this system service has been issued, ResumeOSInterrupts should be issued in the same processing program.
If this system service is re-issued before issuance of ResumeOSInterrupts, only the disable request counter will be incremented (0x1 is added to the disable request counter) and the process to disable the acceptance of interrupts will not be performed (PM*n* bits of PMR are not manipulated).
If the re-issuance of this system service causes the disable request counter (maximum disable request count: 127) to overflow, it will not be handled as an error and no processing will be performed.

Remark 6. The operation when this system service was issued from a processing program but the processing program terminated before ResumeOSInterrupts could be issued will be different according to the definitions in Scalability class "OsScalabilityClass", as follows:

- Scalability class 1 (SC1)
  Correct operation cannot be guaranteed.

- Scalability class 3 (SC3)
  The process will differ as follows depending on the type of processing program.

  [Task]
  Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.
  If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

  [Interrupt service routine (category 2)]
  Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.
  If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter.

  [OS-Application-specific hook routine StartupHook_*OsApplication*]
  Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section.

  [OS-Application-specific hook routine ShutdownHook_*OsApplication*]
  Terminates the critical section.

  [Others]
  Correct operation is not guaranteed.

The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.

[Return values]

None

14.4.3 Resource management

The following shows the system services for resource management provided by the RV850.

Table 14.17 System Services for Resource Management

Name of System Service	Function Overview
GetResource	Acquires a resource.
ReleaseResource	Releases a resource.

GetResource

[Overview]

Acquires a resource.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType GetResource ( ResourceType ResID );
```

[Parameters]

I/O	Parameter	Description
I	`ResourceType` *ResID*`;`	Resource identifier

[Function]

Acquires the target resource (the resource specified in parameter *ResID)*. In this system service, a resource is acquired (the target resource is occupied between issuance of this system service and issuance of ReleaseResource) and the current priority is changed.

Remark 1. The process to change the current priority will be different according to the type of the processing program from which this system service was issued, as follows:

- Task
  The current priority of the current task is changed to Ceiling value "OsResourcePriority" of the resource. As the current priority of the current task is changed, the current task is re-queued at the top of the ready queue corresponding to the changed priority.
  When the ceiling value of the resource is INTPRI*x*, the current priority of the current task is changed to the highest priority (29) and the process to disable the acceptance of interrupts will be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is disabled).
- Interrupt service routine (category 2)
  The current priority of the interrupt service routine is changed to Ceiling value "OsResourcePriority" of the resource and the process to disable the acceptance of interrupts will be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is disabled).

Remark 2. If a single processing program has acquired multiple resources, then the resources must be released in last-in-first-out order (the most recently acquired resource is released first).

Remark 3. When the ceiling value of the acquired resource is higher than the initial priority (Initial priority "OsTaskPriority" or Initial priority "OsIsrPriority") of the processing program that issued this system service and also lower than the current priority of the processing program that issued this system service, the current priority will not be changed and the process to disable the acceptance of interrupts will not be performed. Therefore, even if a task (initial priority: 1) that has already acquired a resource with a ceiling value of 10 acquires a resource with a ceiling value of 5, the current priority remains to be 10.

Remark 4. The operation when an interrupt service routine (category 2) issues this system service but the interrupt service routine processing finishes without issuing ReleaseResource will be different according to the definitions in Scalability class "OsScalabilityClass", as follows:

- Scalability class 1 (SC1)
  Correct operation cannot be guaranteed.

- Scalability class 3 (SC3)
  At the restore process from the interrupt service routine, the resource is released (the occupied target resource is released between issuance of this system service and issuance of ReleaseResource) and the current priority is changed (the current priority of the interrupt service routine is returned to Initial priority "OsIsrPriority").
  If a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the interrupt service routine, the resource will be released, the current priority will be changed, and a hook routine will be called with E_OS_RESOURCE (0x6) used as the parameter.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - Another processing program has already acquired the target resource. - The ceiling value of the target resource is lower than the initial priority (Initial priority "OsTaskPriority" or Initial priority "OsIsrPriority") of the processing program that issued this system service. - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target resource. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ResID* is invalid.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

ReleaseResource

[Overview]

Releases a resource.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType ReleaseResource ( ResourceType ResID );
```

[Parameters]

I/O	Parameter	Description
I	`ResourceType` *ResID*`;`	Resource identifier

[Function]

Releases the target resource (the resource specified in parameter *ResID*). In this system service, a resource is released (the occupied target resource is released between issuance of GetResource and issuance of this system service) and the current priority is changed.

Remark 1. The process to change the current priority will be different according to the type of the processing program from which this system service was issued, as follows:

- Task
  The current priority of the current task is returned to the priority that was set before the resource was acquired. As the current priority of the current task is changed, the current task is re-queued at the top of the ready queue corresponding to the changed priority.
  When the ceiling value of the resource is INTPRI*x*, the current priority of the current task is returned to the priority that was set before the resource was acquired and the process to enable the acceptance of interrupts will be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is enabled).
  If this system service is issued from a task whose Scheduling attribute "OsTaskSchedule" is a preemptive property (FULL), the scheduler is activated when the current priority has been changed.
- Interrupt service routine (category 2)
  The current priority of the interrupt service routine is returned to the priority that was set before the resource was acquired and the process to enable the acceptance of interrupts will be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is enabled).

Remark 2. If a single processing program has acquired multiple resources, then the resources must be released in last-in-first-out order (the most recently acquired resource is released first).

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination

Macro	Numerical Value	Description
E_OS_ACCESS	0x1	Exit with error - Another processing program has already acquired the target resource. - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target resource. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ResID* is invalid.
E_OS_NOFUNC	0x5	Exit with error - Resources were released in an invalid order. - The resource specified in parameter *ResID* has not been acquired by any processing program.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.4 Event management

The following shows the system services for event management provided by the RV850.

Table 14.18 System Services for Event Management

Name of System Service	Function Overview
SetEvent	Sets an event mask.
ClearEvent	Clears an event mask.
GetEvent	Acquires an event mask.
WaitEvent	Confirms an event mask (also shifted to the WAITING state).

SetEvent

[Overview]

Sets an event mask.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType SetEvent ( TaskType TaskID, EventMaskType Mask );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier
I	`EventMaskType` *Mask*`;`	Event mask to be set

[Function]

Sets the event mask specified in parameter *Mask* to the event (32-bit width) assigned to the target task (the extended task specified in parameter *TaskID*).

Remark 1. If the current pattern of the event that is assigned to the target task is B'1100 and the pattern to be set which is specified in parameter *Mask* is B'1010 when this system service is issued, the event mask is set and the current pattern of the event is B'1110.

Remark 2. If multiple events have been assigned to the target task, then the event mask setting process is executed for each event.

Remark 3. When the result of setting the event mask satisfies the request pattern of the target task (WAITING state), then the event mask is set and the state of the task will be manipulated (the target task is shifted from WAITING state to READY state).
The condition of the request pattern is satisfied when comparing "result of setting the event mask" and "request pattern of the target task (the event mask specified in parameter *Mask* when the target task issued WaitEvent)", even a single bit matches (e.g. result of setting is B'0001 and request pattern is B'1111).
When the target task is shifted from WAITING state to READY state, the target task will be queued at the end of the ready queue corresponding to the priority.
If this system service is issued from a task whose Scheduling attribute "OsTaskSchedule" is a preemptive property (FULL), the scheduler is activated when the state of the task has been manipulated (the target task is shifted from WAITING state to READY state).

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination

Macro	Numerical Value	Description
E_OS_ACCESS	0x1	Exit with error - The type of the target task is basic task. - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target task. (Only in SC3) - The OS-Application to which the target task belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *TaskID* is invalid.
E_OS_STATE	0x7	Target task is in SUSPENDED state.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

ClearEvent

[Overview]

Clears an event mask.

[Issue scope]

Tasks (extended)

[Syntax]

```
StatusType ClearEvent ( EventMaskType Mask );
```

[Parameters]

I/O	Parameter	Description
I	`EventMaskType` *Mask*`;`	Event mask to be cleared

[Function]

Clears the current pattern of the event (32-bit width) assigned to the current task (the extended task that issued this system service), using the event mask specified in parameter *Mask*.

Remark 1. If the current pattern of the event that is assigned to the current task is B'1100 and the pattern to be cleared which is specified in parameter *Mask* is B'1010 when this system service is issued, the event mask is cleared and the current pattern of the event is B'0100.

Remark 2. If multiple events have been assigned to the current task, then the event mask clearing process is executed for each event.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Issued from a basic task.
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetEvent

[Overview]

Acquires an event mask.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (PostTaskHook, PreTaskHook, ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetEvent ( TaskType TaskID, EventMaskRefType Event );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier
O	`EventMaskRefType` *Event*`;`	Pointer to the area where an acquired event mask is to be stored

[Function]

Gets the current pattern of the event assigned to the target task (the task specified in parameter *TaskID*), and stores it in the area specified in parameter *Event*.

Remark If multiple events have been assigned to the target task, then the logical sum of the current pattern of each event is stored in the area specified in parameter *Event*.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - Target task is a basic task. - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target task. (Only in SC3) - The OS-Application to which the target task belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *TaskID* is invalid.
E_OS_STATE	0x7	Target task is in SUSPENDED state.
E_OS_PARAM_POINTER	0x12	Specification of parameter *Event* is invalid (NULL pointer).

Macro	Numerical Value	Description
E_OS_ILLEGAL_ADDRERSS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Event*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

WaitEvent

[Overview]

Confirms an event mask (also shifted to the WAITING state).

[Issue scope]

Tasks (extended)

[Syntax]

```
StatusType WaitEvent ( EventMaskType Mask );
```

[Parameters]

I/O	Parameter	Description
I	`EventMaskType` *Mask*`;`	Request pattern

[Function]

Checks whether the current pattern of the event assigned to the current task (the task that issued this system service) meets the conditions of the request pattern specified in parameter *Mask*.

Remark 1. If multiple events have been assigned to the current task, then confirmation processing of the event mask is executed for the logical sum of the current pattern of each event.

Remark 2. The condition of the request pattern is satisfied when comparing "current pattern of the event assigned to the current task" and "request pattern specified in parameter *Mask*", even a single bit matches (e.g. current pattern is B'0001 and request pattern is B'1111).
Accordingly, when the current pattern of the event assigned to the current task is B'1000 and the request pattern is B'0111, the condition is not satisfied.

Remark 3. The processing contents of this system service will be different according to the confirmed result of the event mask, as follows:

- Condition is satisfied
  E_OK (0x0) is returned as the return value of this system service and the current task remains in RUNNING state.
- Condition is not satisfied
  The current task is shifted from RUNNING state to WAITING state.
  As the current task is shifted from RUNNING state to WAITING state, the current task is unlinked from the ready queue corresponding to the priority.
  The scheduler is activated when the state of the task has been manipulated (the current task is shifted from RUNNING state to WAITING state).

Remark 4. If the current task had acquired any internal resources, only in the case of the condition for the confirmed result of the event mask not being satisfied, the state of the task will be manipulated (the current task is shifted from RUNNING state to WAITING state), internal resources will be released, and the current priority will be changed (current priority of current task is returned to the priority that was set before acquiring the resource). After this, the scheduler will be activated.
When the ceiling value of the internal resource is INTPRI*x*, the process to enable the acceptance of interrupts will also be performed (the acceptance of interrupt sources corresponding to Priority "INTPRI0 to ceiling value" is enabled).

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Issued from a basic task.
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_RESOURCE	0x6	Current task has acquired normal resources.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.5 Counter management

The following shows the system services for counter management provided by the RV850.

Table 14.19 System Services for Counter Management

Name of System Service	Function Overview
IncrementCounter	Updates the count value.
GetCounterValue	Gets the current count value.
GetElapsedValue	Gets the current/relative count value.

IncrementCounter

[Overview]

Updates the count value.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType IncrementCounter ( CounterType CounterID );
```

[Parameters]

I/O	Parameter	Description
I	`CounterType` *CounterID*`;`	Counter identifier

[Function]

Increments the target counter (software counter specified by parameter *CounterID*, unit: tick) by 0x1.

Remark 1. If the expiry conditions of the alarm/schedule table associated with the target counter are satisfied, then in addition to incrementing the counter value (adding 0x1 to the counter), the expiry action (e.g. activating tasks) defined during configuration in the Alarm information/Schedule table information is also performed.

Remark 2. When an error (e.g. Activation request count for the task exceeded Maximum activation request count "OsTaskActivation") occurs in the expiry action (e.g. activating tasks) that was performed when this system service was issued, a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) will be called but the return value of this system service will be E_OK (0x0).

Remark 3. If addition causes the counter to overflow (Maximum count value "OsCounterMaxAllowedValue" is exceeded), it will not be handled as an error and 0x0 will be set in the target counter.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target counter. (Only in SC3) - The OS-Application to which the target counter belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Exit with error - Specification of parameter *CounterID* is invalid. - Target counter is a hardware counter.

Macro	Numerical Value	Description
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetCounterValue

[Overview]

Gets the current count value.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType GetCounterValue ( CounterType CounterID, TickRefType Value );
```

[Parameters]

I/O	Parameter	Description
I	`CounterType` *CounterID*`;`	Counter identifier
O	`TickRefType` *Value*`;`	Pointer to the area where the acquired current count value is to be stored

[Function]

Gets the current count value (unit: tick) of the target counter (counter specified by parameter *CounterID*), and stores it in the area specified by parameter *Value*.

Remark The AUTOSAR specifications have a rule (OS531) specifying that the value to be stored in the area specified by parameter *Value* should have a unit complying with the counter type (software counter or hardware counter) specified in parameter *CounterID*. In the RV850, however, the unit is unified to tick, regardless of the counter type.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target counter. (Only in SC3) - The OS-Application to which the target counter belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *CounterID* is invalid.
E_OS_PARAM_POINTER	0x12	Specification of parameter *Value* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Value*. (Only in SC3)

Macro	Numerical Value	Description
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetElapsedValue

[Overview]

Gets the current/relative count value.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType GetElapsedValue ( CounterType CounterID, TickRefType Value, TickRefType
ElapsedValue );
```

[Parameters]

I/O	Parameter	Description
I	`CounterType` *CounterID*`;`	Counter identifier
I/O	`TickRefType` *Value*`;`	Pointer to the area storing the starting count value, and pointer to the area storing the current count value
O	`TickRefType` *ElapsedValue*`;`	Pointer to the area where the acquired relative count value is to be stored

[Function]

Gets the current count value (unit: tick) of the target counter (counter specified by parameter *CounterID*), and stores that current count value in the area specified by parameter *Value*. Also, the difference between the starting count value (unit: tick) specified in parameter *Value* and the current count value (relative count value, unit: tick) is stored in the area specified by parameter *ElapsedValue*.

Remark 1. When this system service is issued, if the current count value of the target counter is 0x5 and the starting count value specified by parameter *Value* is 0x3, then the relative count value that will be stored in the area specified by parameter *ElapsedValue* is 0x2 and the current count value that will be stored in the area specified by parameter *Value* is 0x5.

Remark 2. In the RV850, the number of times a counter has overflowed (Maximum count value "OsCounterMaxAllowedValue" is exceeded) is not held. Therefore, if the target counter overflows during the period starting from the starting count value specified by parameter *Value* until issuance of this system service, the value that will be stored in the area specified by parameter *ElapsedValue* is invalid.

Remark 3. The AUTOSAR specifications have a rule (OS531) specifying that the values to be stored in the areas specified by parameters *Value* and *ElapsedValue* should have a unit complying with the counter type (software counter or hardware counter) specified in parameter *CounterID*. In the RV850, however, the unit is unified to tick, regardless of the counter type.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination

Macro	Numerical Value	Description
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target counter. (Only in SC3) - The OS-Application to which the target counter belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *CounterID* is invalid.
E_OS_VALUE	0x8	The specification by parameter *Value* is invalid (*Value* > Maximum count value "OsCounterMaxAllowedValue").
E_OS_PARAM_POINTER	0x12	The specification by parameter *Value* or parameter *Elapsed-Value* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Value* or the area specified in parameter *ElapsedValue*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.6 Alarm management

The following shows the system services for alarm management provided by the RV850.

Table 14.20 System Services for Alarm Management

Name of System Service	Function Overview
GetAlarmBase	Acquires alarm base information.
GetAlarm	Acquires the remainder count value.
SetRelAlarm	Activates the relative alarm.
SetAbsAlarm	Activates the absolute alarm.
CancelAlarm	Cancels the alarm.

GetAlarmBase

[Overview]

Acquires alarm base information.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (PostTaskHook, PreTaskHook, ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetAlarmBase ( AlarmType AlarmID, AlarmBaseRefType Info );
```

[Parameters]

I/O	Parameter	Description
I	`AlarmType` *AlarmID*`;`	Alarm identifier
O	`AlarmBaseRefType` *Info*`;`	Pointer to the area where the acquired alarm base information is to be stored

[Alarm base information: AlarmBaseRefType]

```
struct _AlarmBaseType {
    TickType maxallowedvalue; /* Maximum count value "OsCounterMaxAllowedValue" */
    TickType mincycle;        /* Minimum cycle value "OsCounterMinCycle" */
    TickType ticksperbase;    /* Basic count value "OsCounterTicksPerBase" */
};

typedef struct _AlarmBaseType AlarmBaseType;
typedef AlarmBaseType *AlarmBaseRefType;
```

[Function]

Gets the alarm base information (Maximum count value "OsCounterMaxAllowedValue", Minimum cycle value "OsCounterMinCycle", and Basic count value "OsCounterTicksPerBase" of the counter associated with the target alarm) for the target alarm (alarm specified in parameter *AlarmID*), and stores it in the area specified by parameter *Info*.

Remark See "14.3.1Alarm base information" for details about the alarm base information (AlarmBaseRefType).

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination

Macro	Numerical Value	Description
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target alarm. (Only in SC3) - The OS-Application to which the target alarm belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *AlarmID* is invalid.
E_OS_PARAM_POINTER	0x12	Specification of parameter *Info* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Info*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetAlarm

[Overview]

Acquires the remainder count value.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (PostTaskHook, PreTaskHook, ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetAlarm ( AlarmType AlarmID, TickRefType Tick );
```

[Parameters]

I/O	Parameter	Description
I	`AlarmType` *AlarmID*`;`	Alarm identifier
O	`TickRefType` *Tick*`;`	Pointer to the area where the acquired remainder count value is to be stored

[Function]

Gets the remainder count value (unit: tick) until expiry conditions of the target alarm (alarm specified in parameter *AlarmID*) are met, and stores it in the area specified by parameter *Tick*.

Remark 1. After this system service is issued, when the count value of the counter associated with the target alarm is updated for the number of values stored in the area specified in parameter *Tick* and the expiry conditions of the target alarm are met, the expiry action (e.g. activating tasks) defined in Expiry action "OsAlarmAction" is performed. When the counter associated with the target alarm is a software counter, after this system service is issued, the expiry conditions of the target alarm are met by issuing IncrementCounter for the number of values stored in the area specified in parameter *Tick*.

Remark 2. When this system service is issued from an interrupt service routine (e.g. interrupt service routine that was called by an interrupt occurring in the middle of the RV850 executing a process associated with IncrementCounter), the acquired remainder count value may be invalid.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target alarm. (Only in SC3) - The OS-Application to which the target alarm belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.

Macro	Numerical Value	Description
E_OS_ID	0x3	Specification of parameter *AlarmID* is invalid.
E_OS_NOFUNC	0x5	The target alarm is in the inactive state.
E_OS_PARAM_POINTER	0x12	Specification of parameter *Tick* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Tick*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

SetRelAlarm

[Overview]

Activates the relative alarm.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType SetRelAlarm ( AlarmType AlarmID, TickType increment, TickType cycle );
```

[Parameters]

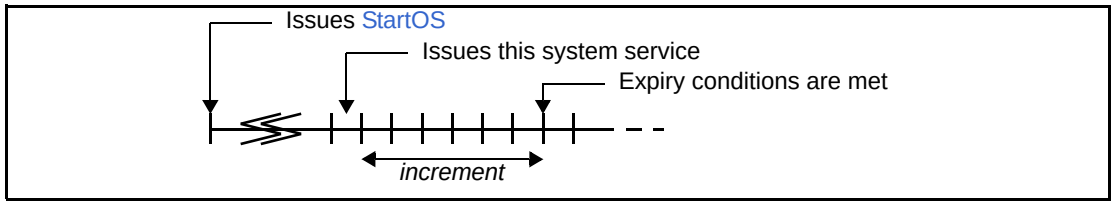
I/O	Parameter	Description
I	`AlarmType` *AlarmID*`;`	Alarm identifier
I	`TickType` *increment*`;`	Relative count value (unit: tick)
I	`TickType` *cycle*`;`	Cycle count value (unit: tick)

[Function]

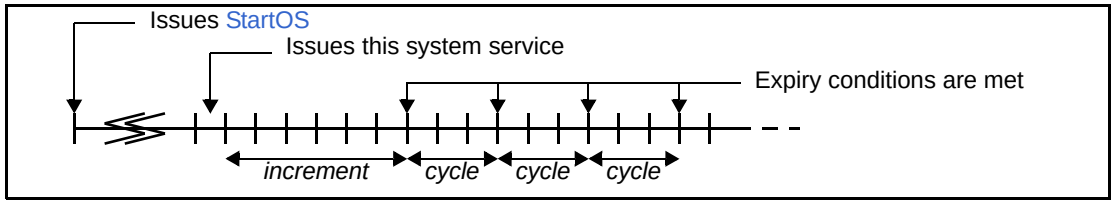
Sets the expiry conditions specified by parameters *increment* and *cycle* in the target alarm (alarm specified in parameter *AlarmID*) and switches the target alarm from the inactive state to the active state.

Remark 1. If 0x0 is set in parameter *cycle*, the target alarm behaves as a one-shot alarm. If a value other than 0x0 is set, then it behaves as a cyclic alarm.

Remark 2. If the target alarm behaves as a one-shot alarm, after this system service is issued, the expiry action is executed when the count value (count value of the counter associated with the target alarm) has been updated for the number of times of "Value specified in parameter *increment* + 1", and the target alarm is shifted from the active state to the inactive state.
The sequence for *increment* = 6 is shown below.



Remark 3. If the target alarm behaves as a cyclic alarm, after this system service is issued, the first-time expiry action is executed when the count value (count value of the counter associated with the target alarm) has been updated for the number of times of "Value specified in parameter *increment* + 1", and after the first-time expiry action has been executed, the expiry action is repeated each time the count value (count value of the counter associated with the target alarm) is updated for the number of times specified in parameter *cycle*.
The sequence for *increment* = 6 and *cycle* = 3 is shown below.



[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target alarm. (Only in SC3) - The OS-Application to which the target alarm belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *AlarmID* is invalid.
E_OS_STATE	0x7	The target alarm is in the active state.
E_OS_VALUE	0x8	Exit with error - Specification of parameter *increment* is invalid (*increment* = 0x0). - Specification of parameter *increment* is invalid (*increment* > Maximum count value "OsCounterMaxAllowedValue"). - Specification of parameter *cycle* is invalid (*cycle* < Minimum cycle value "OsCounterMinCycle"). - Specification of parameter *cycle* is invalid (*cycle* > Maximum count value "OsCounterMaxAllowedValue").
E_OS_DISABLEDINT	0x15	Issued from a critical section.

SetAbsAlarm

[Overview]

Activates the absolute alarm.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType SetAbsAlarm ( AlarmType AlarmID, TickType start, TickType cycle );
```

[Parameters]

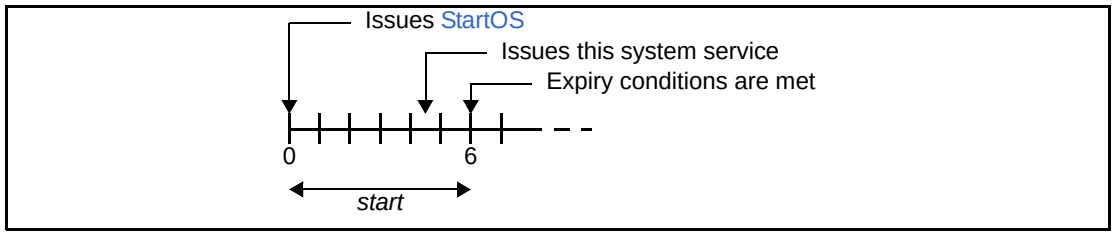
I/O	Parameter	Description
I	`AlarmType` *AlarmID*`;`	Alarm identifier
I	`TickType` *start*`;`	Absolute count value (unit: tick)
I	`TickType` *cycle*`;`	Cycle count value (unit: tick)

[Function]

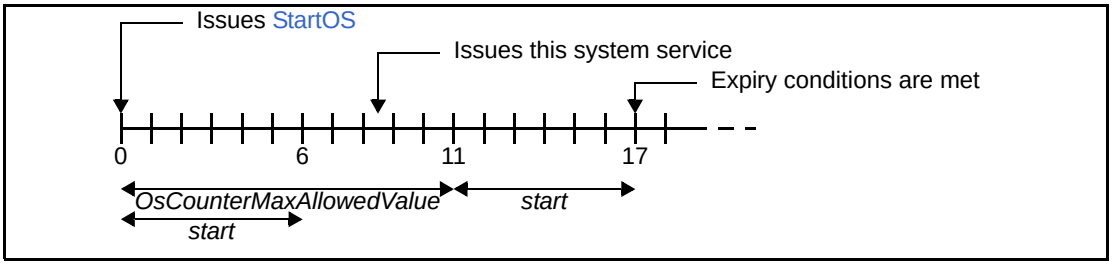
Sets the expiry conditions specified by parameters *start* and *cycle* in the target alarm (alarm specified in parameter *AlarmID*) and switches the target alarm from the inactive state to the active state.

Remark 1. If 0x0 is set in parameter *cycle*, the target alarm behaves as a one-shot alarm. If a value other than 0x0 is set, then it behaves as a cyclic alarm.

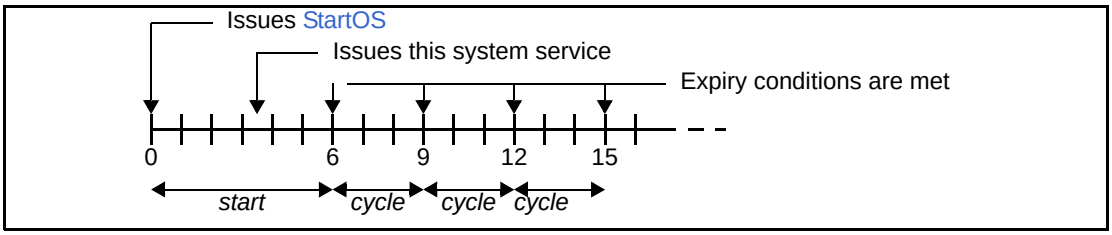
Remark 2. If the target alarm behaves as a one-shot alarm, after this system service is issued, the expiry action is executed when the absolute count value specified in parameter *start* matches the count value (count value of the counter associated with the target alarm), and the target alarm is shifted from the active state to the inactive state.
The sequence for *start* = 6 is shown below.



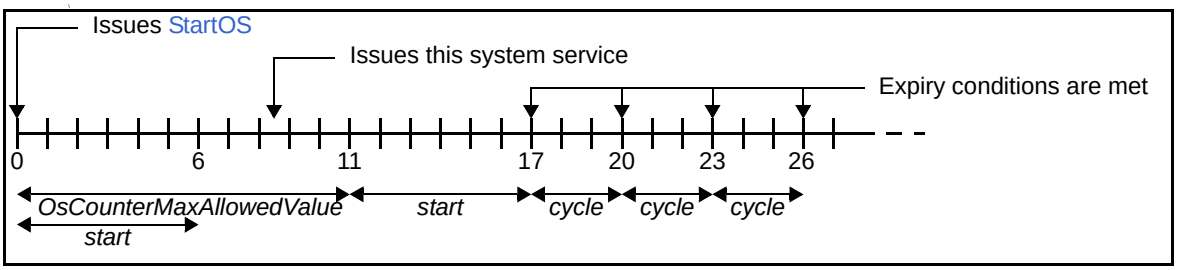
In the RV850, when incrementation of the counter (adding 0x1 to the counter) causes the counter to overflow (Maximum count value "OsCounterMaxAllowedValue" is exceeded), it will not be handled as an error and 0x0 will be set in the target counter.
Therefore, if the value specified in parameter *start* is smaller than the count value (count value of the counter associated with the target alarm), the one-shot alarm will operate as shown below.
The sequence for *start* = 6 and *OsCounterMaxAllowedValue* = 11 is shown below.



Remark 3. If the target alarm behaves as a cyclic alarm, after this system service is issued, the first-time expiry action is executed when the absolute count value specified in parameter *start* matches the current count value (count value of the counter associated with the target alarm), and after the first-time expiry action has been executed, the expiry action is repeated each time the count value (count value of the counter associated with the target alarm) is updated for the number of times specified in parameter *cycle*.
The sequence for *start* = 6 and *cycle* = 3 is shown below.



In the RV850, when incrementation of the counter (adding 0x1 to the counter) causes the counter to overflow (Maximum count value "OsCounterMaxAllowedValue" is exceeded), it will not be handled as an error and 0x0 will be set in the target counter.
Therefore, if the value specified in parameter *start* is smaller than the count value (count value of the counter associated with the target alarm), the cyclic alarm will operate as shown below.
The sequence for *start* = 6, *cycle* = 3, and *OsCounterMaxAllowedValue* = 11 is shown below.



[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target alarm. (Only in SC3) - The OS-Application to which the target alarm belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *AlarmID* is invalid.
E_OS_STATE	0x7	The target alarm is in the active state.

Macro	Numerical Value	Description
E_OS_VALUE	0x8	Exit with error - Specification of parameter *start* is invalid (*start* > Maximum count value "OsCounterMaxAllowedValue"). - Specification of parameter *cycle* is invalid (*cycle* < Minimum cycle value "OsCounterMinCycle"). - Specification of parameter *cycle* is invalid (*cycle* > Maximum count value "OsCounterMaxAllowedValue").
E_OS_DISABLEDINT	0x15	Issued from a critical section.

CancelAlarm

[Overview]

Cancels the alarm.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType CancelAlarm ( AlarmType AlarmID );
```

[Parameters]

I/O	Parameter	Description
I	`AlarmType` *AlarmID*`;`	Alarm identifier

[Function]

Shifts the target alarm (alarm specified in parameter *AlarmID*) from the active state to the inactive state.

Remark 1. Even if the count value of the counter associated with the alarm shifted into the inactive state is updated, the remainder count value will not be decremented until the expiry conditions are met, and no determination will be made as to whether the expiry conditions are met.
Consequently, the expiry conditions for an alarm that has been shifted to the inactive state cannot be met.

Remark 2. An alarm that has been shifted to the inactive state can be shifted to the active state by issuing SetRelAlarm or SetAbsAlarm from a processing program.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target alarm. (Only in SC3) - The OS-Application to which the target alarm belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *AlarmID* is invalid.
E_OS_NOFUNC	0x5	The target alarm is in the inactive state.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.7 Schedule table management

The following shows the system services for schedule table management provided by the RV850.

Table 14.21 System Services for Schedule Table Management

Name of System Service	Function Overview
StartScheduleTableRel	Starts the relative schedule table.
StartScheduleTableAbs	Starts the absolute schedule table.
StopScheduleTable	Stops the schedule table.
NextScheduleTable	Switches the schedule table.
GetScheduleTableStatus	Gets the current state.

StartScheduleTableRel

[Overview]

Starts the relative schedule table.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType StartScheduleTableRel ( ScheduleTableType ScheduleTableID, TickType Offset );
```

[Parameters]

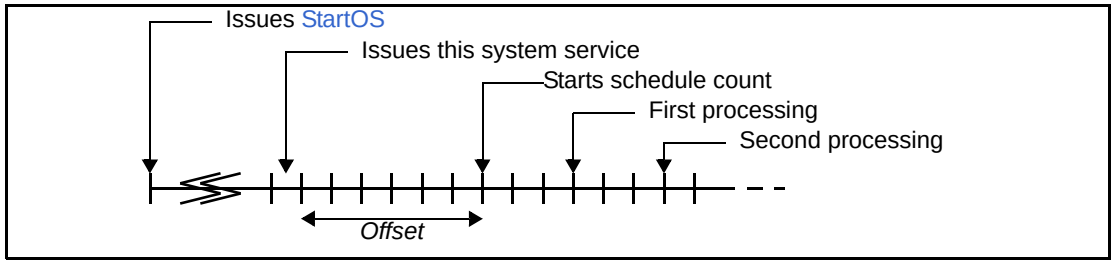
I/O	Parameter	Description
I	`ScheduleTableType` *ScheduleTableID*`;`	Schedule table identifier
I	`TickType` *Offset*`;`	Relative count value

[Function]

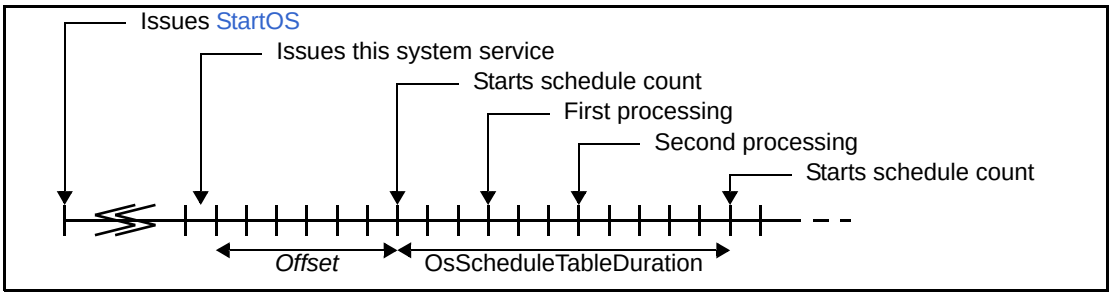
Shifts the target schedule table (the relative schedule table specified in parameter *ScheduleTableID*) from STOPPED state to RUNNING state.

Remark 1. If this system service is issued for a schedule table whose Cyclic property "OsScheduleTableRepeating" is non-cyclic property (FALSE), the schedule table behaves as a one-shot schedule table. If this system service is issued for a schedule table of the cyclic property (TRUE), the schedule table behaves as a cyclic schedule table.

Remark 2. If the target schedule table behaves as a one-shot schedule table, after this system service is issued, the schedule count is started when the count value (count value of the counter associated with the target schedule table) has been updated for the number of times of "Value specified in parameter *Offset* + 1". Then after the final expiry action has been executed, the target schedule table is shifted from RUNNING state to STOPPED state.
The sequence for *Offset* = 6 and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.



Remark 3. If the target schedule table behaves as a cyclic schedule table, after this system service is issued, the schedule count is started when the count value (count value of the counter associated with the target schedule table) has been updated for the number of times of "Value specified in parameter *Offset* + 1". Then, the schedule count is repeated each time the count value is updated for the number of times of the value defined in Schedule count value "OsScheduleTableDuration".
The sequence for *Offset* = 6, OsScheduleTableDuration = 11, and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.



[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target schedule table. (Only in SC3) - The OS-Application to which the target schedule table belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ScheduleTableID* is invalid.
E_OS_STATE	0x7	The target schedule table is in RUNNING state or NEXT state.
E_OS_VALUE	0x8	Exit with error - Specification of parameter *Offset* is invalid (*Offset* <= 0x0). - Specification of parameter *Offset* is invalid (*Offset* > Maximum count value "OsCounterMaxAllowedValue" - Expiry count value "OsScheduleTblExpPointOffset"[Note]).
E_OS_DISABLEDINT	0x15	Issued from a critical section.

Note If more than one Expiry count value "OsScheduleTblExpPointOffset" has been defined for the target schedule table, then Expiry count value "OsScheduleTblExpPointOffset" in the above formula is the minimum value.

StartScheduleTableAbs

[Overview]

Starts the absolute schedule table.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType StartScheduleTableAbs ( ScheduleTableType ScheduleTableID, TickType Start );
```

[Parameters]

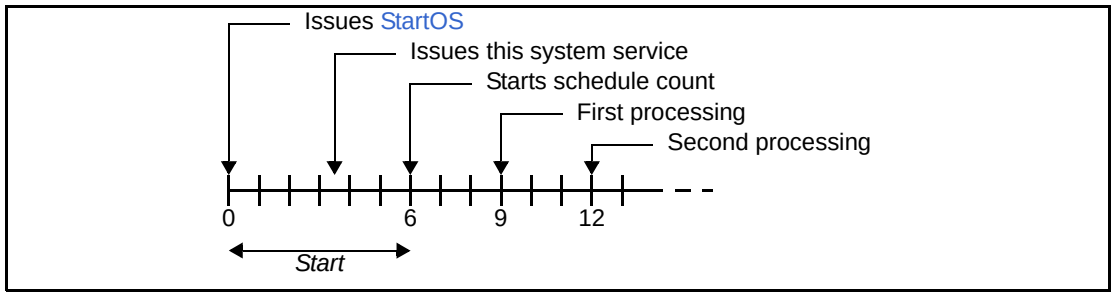
I/O	Parameter	Description
I	`ScheduleTableType` *ScheduleTableID*`;`	Schedule table identifier
I	`TickType` *Start*`;`	Absolute count value

[Function]

Shifts the target schedule table (the absolute schedule table specified in parameter *ScheduleTableID*) from STOPPED state to RUNNING state.

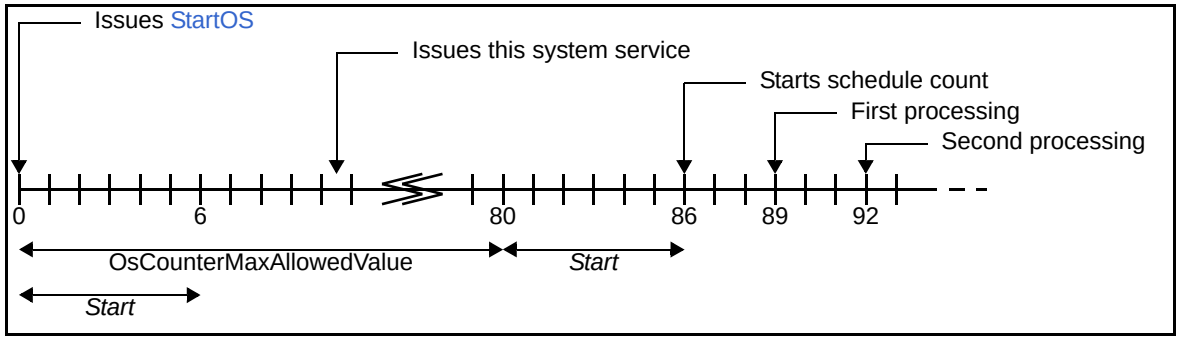
Remark 1. If this system service is issued for a schedule table whose Cyclic property "OsScheduleTableRepeating" is non-cyclic property (FALSE), the schedule table behaves as a one-shot schedule table. If this system service is issued for a schedule table of the cyclic property (TRUE), the schedule table behaves as a cyclic schedule table.

Remark 2. If the target schedule table behaves as a one-shot schedule table, after this system service is issued, the schedule count is started when the absolute count value specified in parameter *Start* matches the count value (count value of the counter associated with the target schedule table). Then after the final expiry action has been executed, the target schedule table is shifted from RUNNING state to STOPPED state. The sequence for *Start* = 6 and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.

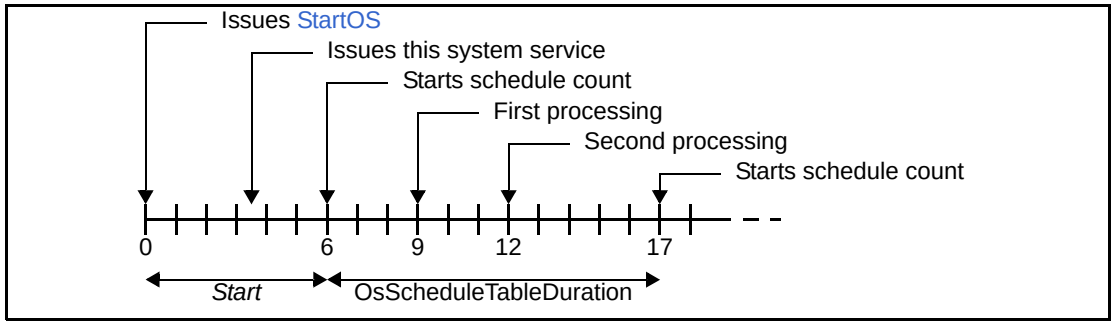


In the RV850, when incrementation of the counter (adding 0x1 to the counter) causes the counter to overflow (Maximum count value "OsCounterMaxAllowedValue" is exceeded), it will not be handled as an error and 0x0 will be set in the target counter.
Therefore, if the value specified in parameter *Start* is smaller than the count value (count value of the counter associated with the target schedule table), the one-shot schedule table will operate as shown below.

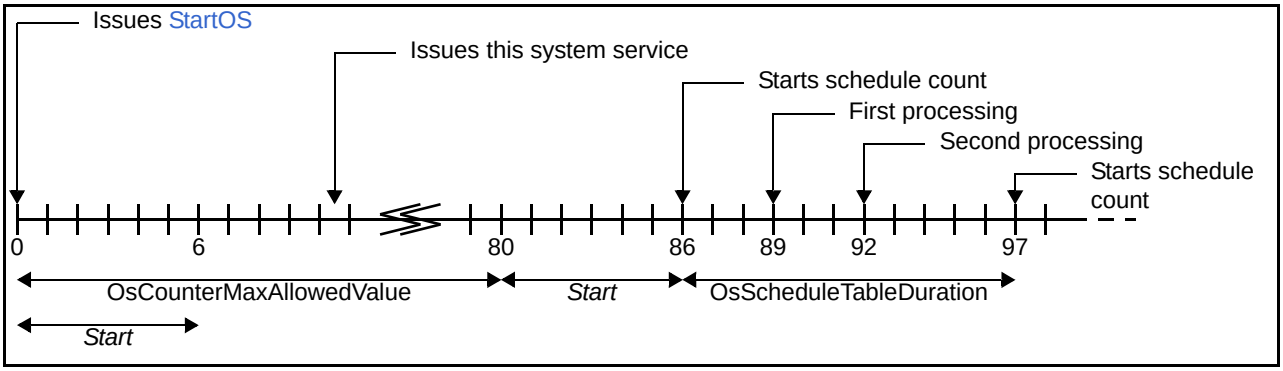
The sequence for *Start* = 6, OsCounterMaxAllowedValue = 80, and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.



Remark 3. If the target schedule table behaves as a cyclic schedule table, after this system service is issued, the schedule count is started when the absolute count value specified in parameter *Start* matches the count value (count value of the counter associated with the target schedule table). Then the schedule count is repeated each time the count value is updated for the number of times of the value defined in Schedule count value "OsScheduleTableDuration".
The sequence for *Start* = 6, OsScheduleTableDuration = 11, and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.



In the RV850, when incrementation of the counter (adding 0x1 to the counter) causes the counter to overflow (Maximum count value "OsCounterMaxAllowedValue" is exceeded), it will not be handled as an error and 0x0 will be set in the target counter.
Therefore, if the value specified in parameter *Start* is smaller than the count value (count value of the counter associated with the target schedule table), the cyclic schedule table will operate as shown below.
The sequence for *Start* = 6, OsCounterMaxAllowedValue = 80, OsScheduleTableDuration = 11, and expiry action count = 2 (OsScheduleTblExpPointOffset = 3 for the first processing and OsScheduleTblExpPointOffset = 6 for the second processing) is shown below.



[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target schedule table. (Only in SC3) - The OS-Application to which the target schedule table belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ScheduleTableID* is invalid.
E_OS_STATE	0x7	The target schedule table is in RUNNING state or NEXT state.
E_OS_VALU	0x8	Specification of parameter *Start* is invalid (*Start* > Maximum count value "OsCounterMaxAllowedValue").
E_OS_DISABLEDINT	0x15	Issued from a critical section.

StopScheduleTable

[Overview]

Stops the schedule table.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType StopScheduleTable ( ScheduleTableType ScheduleTableID );
```

[Parameters]

I/O	Parameter	Description
I	`ScheduleTableType` *ScheduleTableID*`;`	Schedule table identifier

[Function]

Shifts the target schedule table (the schedule table specified in parameter *ScheduleTableID*) from RUNNING state or NEXT state to STOPPED state.

Remark If the target schedule table was the schedule table before the switch (the schedule table specified in parameter *ScheduleTableID_From* when issuing NextScheduleTable), then both the target schedule table and the schedule table specified in parameter *ScheduleTableID_To* upon issuance of NextScheduleTable are shifted to STOPPED state.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target schedule table. (Only in SC3) - The OS-Application to which the target schedule table belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ScheduleTableID* is invalid.
E_OS_NOFUNC	0x5	The target schedule table is in STOPPED state.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

NextScheduleTable

[Overview]

Switches the schedule table.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType NextScheduleTable ( ScheduleTableType ScheduleTableID_From, ScheduleTableType ScheduleTableID_To );
```

[Parameters]

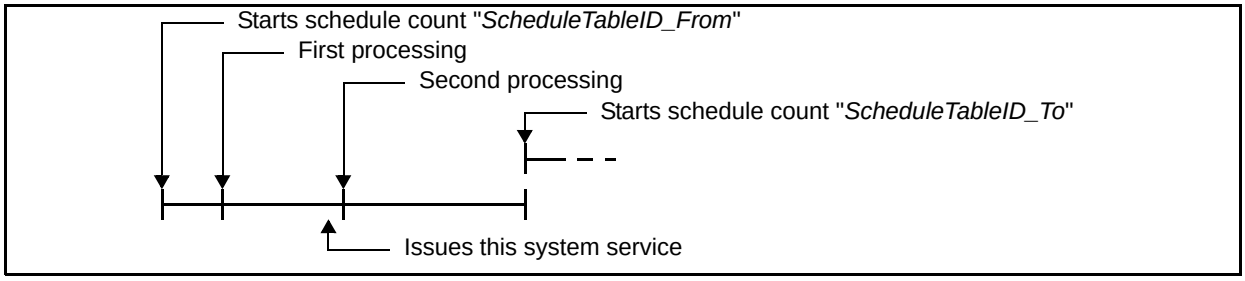
I/O	Parameter	Description
I	`ScheduleTableType` *ScheduleTableID_From*`;`	Schedule table identifier
I	`ScheduleTableType` *ScheduleTableID_To*`;`	Schedule table identifier

[Function]

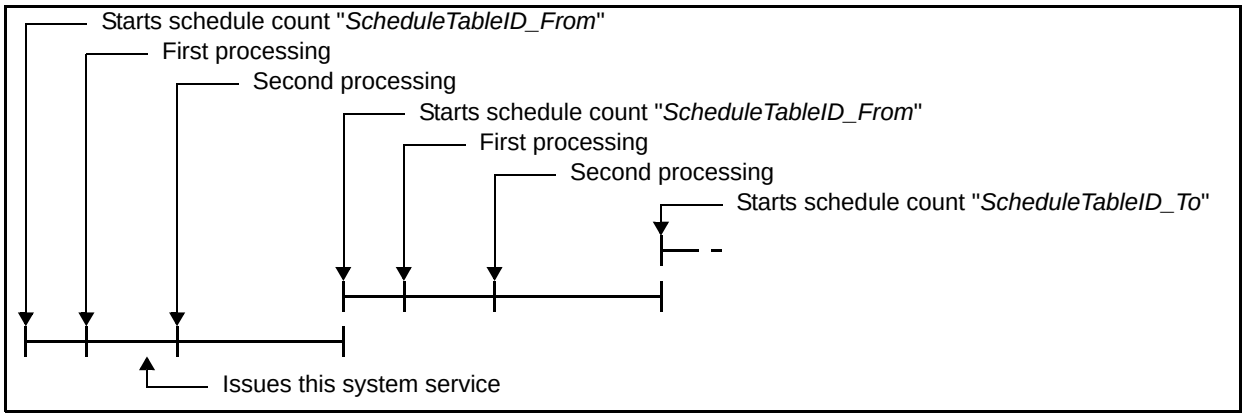
Switches from the schedule table specified by parameter *ScheduleTableID_From* to the schedule table specified by parameter *ScheduleTableID_To*.

Note that the timing of switching schedule tables depends on the timing with which this system service is issued.

- This system service is issued before the final expiry action is executed.
  The switch is made when the schedule count of schedule table *ScheduleTableID_From* is complete.
  Consequently, as shown in the figure below, if two expiry actions have been defined for schedule table *ScheduleTableID_From*, and this system service was issued before the second expiry action (final expiry action) was executed, then the shift will be made to schedule table *ScheduleTableID_To* when the schedule count of schedule table *ScheduleTableID_From* is complete.



- This system service is issued after the final expiry action is executed.
  When the schedule count of schedule table *ScheduleTableID_From* is complete, the schedule count of schedule table *ScheduleTableID_From* is started again, and after the final expiry action is executed, the switch to schedule table *ScheduleTableID_To* is made.
  Consequently, as shown in the figure below, if two expiry actions have been defined for schedule table *ScheduleTableID_From*, and this system service is issued after the second expiry action (final expiry action) is executed, then when the schedule count of schedule table *ScheduleTableID_From* is complete, the schedule count *ScheduleTableID_From* of the schedule table is started again, and after that second schedule count of schedule table *ScheduleTableID_From* is complete, the switch is made to schedule table *ScheduleTableID_To*.



Remark 1. The schedule table specified by parameter *ScheduleTableID_From* is changed to STOPPED state after the schedule tables are switched.
After the shift to NEXT state via issuance of this system service, the schedule table specified by parameter *ScheduleTableID_To* is changed to RUNNING state after the schedule tables are switched.

Remark 2. If this system service is issued again between issuance of this system service and the switch of schedule tables, then the schedule table specified by parameter *ScheduleTableID_To* at the earlier issuance of this system service will be changed from NEXT state to STOPPED state.

Remark 3. The AUTOSAR specifications do not have rules regarding Cyclic property "OsScheduleTableRepeating" of schedule tables that can be specified by parameters *ScheduleTableID_From* and *ScheduleTableID_To*.
In the RV850, however, Cyclic property "OsScheduleTableRepeating" of schedule tables that can be specified by the above parameters is limited to "Assign cyclic property (TRUE)". If a schedule table with a property other than cyclic property is specified, error status E_OS_ID is returned.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target schedule table. (Only in SC3) - The OS-Application to which the target schedule table belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.

Macro	Numerical Value	Description
E_OS_ID	0x3	Exit with error - Specification of parameter *ScheduleTableID_From* is invalid. - Specification of parameter *ScheduleTableID_To* is invalid. - Cyclic property "OsScheduleTableRepeating" of the schedule table specified by parameter *ScheduleTableID_From* is "Do not assign cyclic property (FALSE)". - Cyclic property "OsScheduleTableRepeating" of the schedule table specified by parameter *ScheduleTableID_To* is "Do not assign cyclic property (FALSE)". - The counter associated with the schedule table specified by parameter *ScheduleTableID_From* is different from the counter associated with the schedule table specified by parameter *ScheduleTableID_To*.
E_OS_NOFUNC	0x5	The target schedule table specified by parameter *ScheduleTableID_From* is in STOPPED state or NEXT state.
E_OS_STATE	0x7	The target schedule table specified by parameter *ScheduleTableID_To* is in NEXT state or RUNNING state.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetScheduleTableStatus

[Overview]

Gets the current state.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType GetScheduleTableStatus ( ScheduleTableType ScheduleTableID, ScheduleTableSta-
tusRefType ScheduleStatus );
```

[Parameters]

I/O	Parameter	Description
I	`ScheduleTableType` *ScheduleTableID*`;`	Schedule table identifier
O	`ScheduleTableStatusRefType` *ScheduleStatus*`;`	Pointer to the area where the acquired current state is to be stored

[Function]

Gets the current state of the target schedule table (the schedule table specified in parameter *ScheduleTableID*), and stores it in the area specified in parameter *ScheduleStatus*.

The value stored in parameter *ScheduleStatus* depends on the current state type, as shown below.

Macro	Numerical Value	Description
SCHEDULETABLE_STOPPED	0x0	STOPPED state
SCHEDULETABLE_NEXT	0x1	NEXT state
SCHEDULETABLE_RUNNING	0x4	RUNNING state

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	Exit with error - The OS-Application to which the processing program that issued this system service belongs does not have access privileges for the target schedule table. (Only in SC3) - The OS-Application to which the target schedule table belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state. (Only in SC3)
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *ScheduleTableID* is invalid.

Macro	Numerical Value	Description
E_OS_PARAM_POINTER	0x12	Specification of parameter *ScheduleStatus* is invalid (NULL pointer).
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *ScheduleStatus*. (Only in SC3)
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.8 OS-Application management

The following shows the system services for OS-Application management provided by the RV850.

Table 14.22 System Services for OS-Application Management

Name of System Service	Function Overview
GetApplicationID	Gets the OS-Application identifier.
GetISRID	Gets the interrupt service routine identifier.
CallTrustedFunction	Calls a trusted function.
CheckISRMemoryAccess	Checks access privileges (interrupt service routine).
CheckTaskMemoryAccess	Checks access privileges (task).
CheckObjectAccess	Checks access privileges (object).
CheckObjectOwnership	Gets the OS-Application identifier.
TerminateApplication	Terminates the OS-Application.
AllowAccess	Activates the OS-Application.
GetApplicationState	Gets the current state.

GetApplicationID

[Overview]

Gets the OS-Application identifier.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook), OS-Application-specific hook routines (StartupHook_*OsApplication*, ShutdownHook_*OsApplication*, ErrorHook_*OsApplication*)

[Syntax]

```
ApplicationType GetApplicationID ( void );
```

[Parameters]

None

[Function]

Gets the identifier of the target OS-Application (the OS-Application to which the processing program that issued this system service belongs), and returns the OS-Application identifier.

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. When this system service is issued from a common hook routine (PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook) or an OS-Application-specific hook routine (ErrorHook_*OsApplication*), the identifier of the OS-Application to which the processing program that called the hook routine belongs is returned.

Remark 3. The correspondence between the acquired numerical value and Identifier "OsApplication" is defined in the SIT file output from the configurator.

[Return values]

Macro	Numerical Value	Description
-	Other than 0x7FFF	Acquired OS-Application identifier
INVALID_OSAPPLICATION	0x7FFF	Issued from a processing program outside the scope of issue.

Remark If this system service is issued from the common hook routine (ProtectionHook) called from a processing program that is neither a task nor an interrupt service routine (category 2), INVALID_OSAPPLICATION (0x7FFF) will be returned.

GetISRID

[Overview]

Gets the interrupt service routine identifier.

[Issue scope]

Interrupt service routines (category 2), common hook routines (ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
ISRType GetISRID ( void );
```

[Parameters]

None

[Function]

Gets the identifier of the target interrupt service routine (the interrupt service routine that issued this system service, or the interrupt service routine that called the hook routine that issued this system service), and returns the interrupt service routine identifier.

Remark 1. If this system service is issued from a hook routine called from a task, INVALID_ISR (0x7FFF) will be returned.

Remark 2. The correspondence between the acquired numerical value and Identifier "OsIsr" is defined in the SIT file output from the configurator.

[Return values]

Macro	Numerical Value	Description
-	Other than 0x7FFF	Acquired interrupt service routine identifier
INVALID_ISR	0x7FFF	Issued from a processing program outside the scope of issue.

Remark If this system service is issued from the common hook routine (ErrorHook or ProtectionHook) called by a task or is issued from the OS-Application-specific hook routine (ErrorHook_*OsApplication*), INVALID_ISR (0x7FFF) will be returned.

CallTrustedFunction

[Overview]

Calls a trusted function.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType CallTrustedFunction ( TrustedFunctionIndexType FunctionIndex, TrustedFunc-
tionParameterRefType FunctionParams );
```

[Parameters]

I/O	Parameter	Description
I	`TrustedFunctionIndexType` *FunctionIndex*`;`	Trusted function identifier
I	`TrustedFunctionParameterRefType` *FunctionParams*`;`	Pointer to the area where inherited data is stored.

[Function]

Calls the target trusted function (the trusted function specified in parameter *FunctionIndex*).

When a trusted function is called from processing of this system service, the value specified in parameter FunctionIndex is passed as the first argument to the trusted function and the value (pointer) specified in parameter FunctionParams is passed as the second argument to the trusted function.

For details on the data (parameters) inherited to the trusted function, see "10.2.3 Inherited data of trusted functions"

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. When calling a trusted function that has no arguments, set parameter *FunctionParams* to NULL.

Remark 3. Trusted functions operate in supervisor mode; if this system service is issued from a processing program that belongs to a non-trusted OS-Application, the mode switching processing (transition from user mode to supervisor mode) is executed.

Remark 4. This system service does not check the validity of the inherited data indicated by parameter Function-Params. When the validity of the inherited data needs to be checked, check it in the trusted function called by issuing this system service.

Remark 5. See "10.2Trusted Functions" for details on trusted functions.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	The OS-Application to which the target trusted function belongs is in APPLICATION_RESTARTING state or APPLICATION_TERMINATED state.

Macro	Numerical Value	Description
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_SERVICEID	0x11	Specification of parameter *FunctionIndex* is invalid.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

CheckISRMemoryAccess

[Overview]

Checks access privileges (interrupt service routine).

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
AccessType CheckISRMemoryAccess ( ISRType ISRID, MemoryStartAddressType Address, MemorySizeType Size );
```

[Parameters]

I/O	Parameter	Description
I	`ISRType` *ISRID*`;`	Interrupt service routine identifier
I	`MemoryStartAddressType` *Address*`;`	Start address of the memory area
I	`MemorySizeType` *Size*`;`	Size of the memory area (in bytes)

[Function]

Checks whether the target interrupt service routine (the interrupt service routine specified in parameter *ISRID*) has access privileges for the target memory area (the memory area specified by parameters *Address* and *Size*) and whether the target memory area is a stack area, and returns a value indicating the results.

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. The return value from this system service is the valid access privileges for the entire target memory area. Consequently, if part of the memory area is non-writable, the return value will indicate that the entire target memory area is non-writable.

Remark 3. A processing program can issue this system service regardless of whether the program has an access privilege for the OS-Application to which the target interrupt service routine belongs.

Remark 4. A processing program can issue this system service regardless of the state of the OS-Application to which the target interrupt service routine belongs.

[Return values]

Macro	Numerical Value	Description
T_u2_NOACCESS	0x0	Normal termination - No access privileges Exit with error - Issued from a processing program outside the scope of issue. - Specification of parameter *ISRID* is invalid. - Issued from a critical section.
T_u2_EXECUTABLE	0x2	Executable
T_u2_READABLE	0x4	Readable
T_u2_WRITEABLE	0x8	Writable
T_u2_STACKSPACE	0x10	Stack area

Remark The ORed value of the check results (the type of access privileges owned by the target interrupt service routine and whether the area is a stack area) is returned. When the type of access privileges owned by the target interrupt service routine is executable, readable, and non-writable and the target memory area is a non-stack area, the return value will be "0x6".

CheckTaskMemoryAccess

[Overview]

Checks access privileges (task).

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
AccessType CheckTaskMemoryAccess ( TaskType TaskID, MemoryStartAddressType Address, MemorySizeType Size );
```

[Parameters]

I/O	Parameter	Description
I	`TaskType` *TaskID*`;`	Task identifier
I	`MemoryStartAddressType` *Address*`;`	Start address of the memory area
I	`MemorySizeType` *Size*`;`	Size of the memory area (in bytes)

[Function]

Checks whether the target task (the task specified in parameter *TaskID*) has access privileges for the target memory area (the memory area specified by parameters *Address* and *Size*) and whether the target memory area is a stack area, and returns a value indicating the results.

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. The return value from this system service is the valid access privileges for the entire target memory area. Consequently, if part of the memory area is non-writable, the return value will indicate that the entire target memory area is non-writable.

Remark 3. A processing program can issue this system service regardless of whether the program has an access privilege for the OS-Application to which the target task belongs.

Remark 4. A processing program can issue this system service regardless of the state of the OS-Application to which the target task belongs.

[Return values]

Macro	Numerical Value	Description
T_u2_NOACCESS	0x0	Normal termination - No access privileges Exit with error - Issued from a processing program outside the scope of issue. - Specification of parameter *TaskID* is invalid. - Issued from a critical section.
T_u2_EXECUTABLE	0x2	Executable
T_u2_READABLE	0x4	Readable
T_u2_WRITEABLE	0x8	Writable
T_u2_STACKSPACE	0x10	Stack area

Remark The ORed value of the check results (the type of access privileges owned by the target task and whether the area is a stack area) is returned. When the type of access privileges owned by the target task is non-executable, readable, and writable and the target memory area is a stack area, the return value will be "0x1C".

CheckObjectAccess

[Overview]

Checks access privileges (object).

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication)*

[Syntax]

```
ObjectAccessType CheckObjectAccess ( ApplicationType ApplID, ObjectTypeType ObjectType,
<Object>Type ObjectID );
```

Remark The variable type (<*Object*>Type) of parameter *ObjectID* will differ depending on the type of object specified (e.g. TaskType, ISRType, or AlarmType).

[Parameters]

I/O	Parameter	Description
I	`ApplicationType` *ApplID*`;`	OS-Application identifier
I	`ObjectTypeType` *ObjectType*`;`	Object type
I	<*Object*>`Type` *ObjectID*`;`	Object identifier

[Function]

Checks whether objects (tasks, interrupt service routines, alarms, etc.) belonging to the target OS-Application (the OS-Application specified in parameter *ApplID*) have access privileges for the object specified by parameters *ObjectType* and *ObjectID*, and returns a value indicating the result.

Specify parameter *ObjectType* as follows, depending on the object type.

Macro	Numerical Value	Description
OBJECT_TASK	0x1	Task
OBJECT_ISR	0x2	Interrupt service routine
OBJECT_ALARM	0x3	Alarm
OBJECT_RESOURCE	0x4	Resource
OBJECT_COUNTER	0x5	Counter
OBJECT_SCHEDULETABLE	0x6	Schedule table

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. A processing program can issue this system service regardless of whether the program has an access privilege for the OS-Application to which the target object belongs.

Remark 3. A processing program can issue this system service regardless of the state of the OS-Application to which the target object belongs.

[Return values]

Macro	Numerical Value	Description
NO_ACCESS	0x0	Normal termination - No access privileges Exit with error - Issued from a processing program outside the scope of issue. - Specification of parameter *AppliID* is invalid. - Specification of parameter *ObjectType* is invalid. - Specification of parameter *ObjectID* is invalid. - Issued from a critical section.
ACCESS	0x1	Has access privileges.

CheckObjectOwnership

[Overview]

Gets the OS-Application identifier.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (ErrorHook, ProtectionHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
ApplicationType CheckObjectOwnership ( ObjectTypeType ObjectType, <Object>Type ObjectID );
```

Remark The variable type (<*Object*>Type) of parameter *ObjectID* will differ depending on the type of object specified (e.g. TaskType, ISRType, or AlarmType).

[Parameters]

I/O	Parameter	Description
I	`ObjectTypeType` *ObjectType*`;`	Object type
I	<*Object*>`Type` *ObjectID*`;`	Object identifier

[Function]

Gets the identifier of the OS-Application to which the target object (the object specified by parameters *ObjectType* and *ObjectID*) belongs, and returns a value indicating the result.

Specify parameter *ObjectType* as follows, depending on the object type.

Macro	Numerical Value	Description
OBJECT_TASK	0x1	Task
OBJECT_ISR	0x2	Interrupt service routine
OBJECT_ALARM	0x3	Alarm
OBJECT_RESOURCE	0x4	Resource
OBJECT_COUNTER	0x5	Counter
OBJECT_SCHEDULETABLE	0x6	Schedule table

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. A processing program can issue this system service regardless of the state of the OS-Application to which the target object belongs.

[Return values]

Macro	Numerical Value	Description
-	Other than 0x7FFF	Acquired OS-Application identifier
INVALID_OSAPPLICATION	0x7FFF	Exit with error - The processing program that issued this system service has no access privileges for the OS-Application to which the target object belongs. - Issued from a processing program outside the scope of issue. - Specification of parameter *ObjectType* is invalid. - Specification of parameter *ObjectID* is invalid. - Issued from a critical section.

TerminateApplication

[Overview]

Terminates the OS-Application.

[Issue scope]

Tasks, interrupt service routines (category 2), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
StatusType TerminateApplication ( ApplicationType Application, RestartType RestartOption );
```

[Parameters]

I/O	Parameter	Description
I	`ApplicationType` *Application*`;`	OS-Application identifier
I	`RestartType` *RestartOption*`;`	Restart option NO_RESTART (0x0): Terminates the OS-Application. RESTART (0x1): Terminates and restarts the OS-Application.

[Function]

Executes the operation specified in parameter *RestartOption* for the target OS-Application (the OS-Application specified in parameter *Application*).

Remark 1. This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. The operation to be executed for the target OS differs as follows depending on the setting in parameter *RestartOption*.

[NO_RESTART (0x0)]
This system service executes the following operation as OS-application termination processing.

- Shifts the tasks belonging to the target OS-Application to SUSPENDED state.
- Disables acceptance of interrupts corresponding to any interrupt service routine belonging to the target OS-Application (manipulates EI level interrupt mask register IMR*m*).
- Releases the resources that have been acquired by processing programs belonging to the target OS-Application.
- Shifts the alarms belonging to the target OS-Application to inactive state.
- Shifts the schedule tables belonging to the target OS-Application to STOPPED state.
- Shifts the target OS-Application to APPLICATION_TERMINATED state.

[RESTART (0x1)]
This system service executes the following operation after executing the same operation as when NO_RESTART (0x0) is specified.

- Shifts the target task from SUSPENDED state to READY state when Task identifier "OsRestartTask" is specified.
- Shifts the target OS-Application from APPLICATION_TERMINATED state to APPLICATION_RESTARTING state.

The OS-Application placed in APPLICATION_RESTARTING state by issuing this system service is shifted to APPLICATION_ACCESSIBLE state when the task specified by Task identifier "OsRestartTask" issues AllowAccess.
Acceptance of the interrupts disabled by issuing this system service is enabled when the task specified by Task identifier "OsRestartTask" issues InitApplicationInterrupts.

Remark 3. If there is a task in RUNNING state when this system service is issued, the common hook routine (PostTaskHook) is called before that task is shifted to SUSPENDED state by the OS-Application termination processing

Remark 4. When a processing program that belongs to a non-trusted OS-Application issues this system service with parameter *Application* set to "an OS-Application to which the processing program that issues this system service does not belong", E_OS_ACCESS (0x1) will be returned.

Remark 5. When ErrorHook_*OsApplication* issues this system service with parameter *Application* set to "an OS-Application to which ErrorHook_*OsApplication* that issues this system service does not belong", E_OS_CALLEVEL (0x2) will be returned.

Remark 6. This system service does not clear the interrupts whose acceptance is suspended.
Therefore, if these suspended interrupts need to be cleared when the OS-Application is restarted, the user should execute the interrupt clear processing before enabling acceptance of interrupts.

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_ACCESS	0x1	The OS-Application to which the processing program that issues this system service belongs is non-trusted.
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *Application* is invalid.
E_OS_STATE	0x7	Exit with error - The target is an OS-Application in APPLICATION_RESTARTING state, to which the processing program that issued this system service does not belong. - An invalid setting is made for restart option RESTART for an OS-Application in APPLICATION_RESTARTING state, to which the processing program that issued this system service belongs. - The target OS-Application is in APPLICATION_TERMINATED state.
E_OS_VALUE	0x8	Specification of parameter *RestartOption* is invalid.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

AllowAccess

[Overview]

Activates the OS-Application.

[Issue scope]

Tasks, interrupt service routines (category 2)

[Syntax]

```
StatusType AllowAccess ( void );
```

[Parameters]

None

[Function]

Shifts the target OS-Application (the OS-Application to which the processing program that issued this system service belongs) from APPLICATION_RESTARTING state to APPLICATION_ACCESSIBLE state.

Remark This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_STATE	0x7	The OS-Application to which the processing program that issued this system service belongs is in APPLICATION_ACCESSIBLE state or APPLICATION_TERMINATED state.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

GetApplicationState

[Overview]

Gets the current state.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, ErrorHook, ProtectionHook), OS-Application-specific hook routines (StartupHook_*OsApplication*, ShutdownHook_*OsApplication*, ErrorHook_*OsApplication*)

[Syntax]

```
StatusType GetApplicationState ( ApplicationType Application, ApplicationStateRefType
Value );
```

[Parameters]

I/O	Parameter	Description
I	`ApplicationType` *Application*`;`	OS-Application identifier
O	`ApplicationStateRefType` *Value*`;`	Pointer to the area where the acquired current state is to be stored.

[Function]

Gets the current state of the target OS-Application (the OS-Application specified in parameter *Application*) and stores it in the area specified in parameter *Value*.

Parameter *Value* is set as follows, depending on the current state.

Macro	Numerical Value	Description
APPLICATION_ACCESSIBLE	0x0	APPLICATION_ACCESSIBLE state
APPLICATION_RESTARTING	0x1	APPLICATION_RESTARTING state
APPLICATION_TERMINATED	0x2	APPLICATION_TERMINATED state

Remark This system service can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

[Return values]

Macro	Numerical Value	Description
E_OK	0x0	Normal termination
E_OS_CALLEVEL	0x2	Issued from a processing program outside the scope of issue.
E_OS_ID	0x3	Specification of parameter *Application* is invalid.
E_OS_PARAM_POINTER	0x12	Specification of parameter *Value* is invalid (NULL pointer).

Macro	Numerical Value	Description
E_OS_ILLEGAL_ADDRESS	0x13	The processing program that issued this system service has no access privileges for the area specified in parameter *Value*.
E_OS_DISABLEDINT	0x15	Issued from a critical section.

14.4.9 OS execution management

The following shows the system services for OS execution management provided by the RV850.

Table 14.23 System Services for OS Execution Management

Name of System Service	Function Overview
StartOS	Starts RV850.
ShutdownOS	Terminates RV850.
GetActiveApplicationMode	Acquires the application mode.

StartOS

[Overview]

Starts RV850.

[Issue scope]

Boot process

[Syntax]

```
void StartOS ( AppModeType Mode );
```

[Parameters]

I/O	Parameter	Description
I	`AppModeType` *Mode*`;`	Application mode Name: Normal application mode OSDEFAULTAPPMODE: Default application mode

[Function]

Executes the startup processing for the RV850.

Remark 1. The startup processes executed by this system service are given below.

- Disables acceptance of interrupts
  Manipulates the ID bit of the program status word (PSW)
- Initializes the OS reserved resources
- Calls the kernel initialization module
  Creates and registers objects
  Activates objects
- Enables acceptance of interrupts in category 1
  Manipulates the PM*n* bits of the priority mask register (PMR)
  Manipulates the ID bit of the program status word (PSW)
- Calls hook routines
  Calls StartupHook
  Calls StartupHook_*OsApplication*
- Activates the scheduler
- Enables acceptance of interrupts
  Manipulates the PM*n* bits of the priority mask register (PMR)

Remark 2. Since this system service manipulates system registers, it must be issued in supervisor mode (the UM bit in PSW is 0). Also, this system service must be issued after setting of the system protection identifier (SPID bit of MCFG0 register) (Remark 6. in "4.2 Boot Process") is complete.

Remark 3. When the value specified in parameter *Mode* matches the value defined in Application mode "OsTaskAppModeRef" for a task, the task is shifted from SUSPENDED state to READY state when this system service is issued.

Remark 4. When the value specified in parameter *Mode* matches the value defined in Application mode "OsTaskAppModeRef" for an alarm, the alarm is shifted from inactive state to active state when this system service is issued.

Remark 5. When the value specified in parameter *Mode* matches the value defined in Application mode "OsAlarmAppModeRef" for a schedule table, the schedule table is shifted from STOPPED state to RUNNING state when this system service is issued.

Remark 6. This system service does not manipulate the EI level interrupt mask register (IMR*m*).
Therefore, the user should manipulate IMR*m* in the processing program after StartupHook to enable acceptance of interrupts.

[Return values]

None

ShutdownOS

[Overview]

Terminates RV850.

[Issue scope]

Tasks, interrupt service routines (category 2), common hook routines (StartupHook, ErrorHook), OS-Application-specific hook routines (StartupHook_*OsApplication*, ErrorHook_*OsApplication*)

[Syntax]

```
void ShutdownOS ( StatusType Error );
```

[Parameters]

I/O	Parameter	Description
I	`StatusType` *Error*`;`	Inherited data

[Function]

Executes the termination processing for the RV850.

Remark 1. The termination processes executed by this system service are given below.

- Disables acceptance of interrupts
  Manipulates the ID bit of the program status word (PSW)
  Manipulates the PM*n* bits of the priority mask register (PMR)
  Manipulates the EI level interrupt mask register (IMR*m*)
- Calls hook routines
  Calls ShutdownHook_*OsApplication* using the value specified in parameter *Error* as inherited data
  Calls ShutdownHook using the value specified in parameter *Error* as inherited data
- Calls an empty process
  Executes an endless loop of an empty process (shifts RV850 into a pseudo-HALT state)

Remark 2. Execution returns from the empty process only when a hardware reset occurs.

Remark 3. If this system service is issued from a processing program belonging to a non-trusted OS-Application, it will not be handled as an error and no processing will be performed.

Remark 4. If this system service is issued from a processing program outside the valid scope of issue, it will not be handled as an error and no processing will be performed.

Remark 5. In the RV850, when the common hook routine (ProtectionHook) is not defined and if a stack overflow is detected, this system service is issued with parameter *Error* set to E_OS_STACKFAULT (0x16).

Remark 6. In the RV850, if an interrupt that is not defined in the Interrupt service routine information occurs, this system service is issued with parameter *Error* set to E_OS_SYS_ILLEGAL_EXCEPTION (0x1F).

Remark 7. The AUTOSAR specifications do not prescribe whether common hook routine PostTaskHook is called when Scalability class "OsScalabilityClass" is set to scalability class 1 (SC1).
In the RV850, common hook routine PostTaskHook is not called from this system service regardless of the definition of Scalability class "OsScalabilityClass".

[Return values]

None

GetActiveApplicationMode

[Overview]

Acquires the application mode.

[Issue scope]

Tasks, interrupt service routines (category 2), alarm callback, common hook routines, OS-Application-specific hook routines, critical sections

[Syntax]

```
AppModeType GetActiveApplicationMode ( void );
```

[Parameters]

None

[Function]

Gets the application mode that was specified in parameter *OsAppMode* in StartOS, and returns the application mode.

[Return values]

Macro	Numerical Value	Description
-	0x0 to 0x7E	Acquired application mode

Remark If an illegal value is specified in parameter *OsAppMode* for StartOS, a value not shown above may be returned.

14.4.10 Utility functions

The following shows the utility functions provided by the RV850.

Table 14.24 Utility Functions

Name of Utility Functions	Function Overview
InitApplicationInterrupts	Enables acceptance of interrupts.
_kernel_fv0_InitializeIntService	Enables the issuing of system services for interrupt handling.
OSIllegalException_SystemRegister_ExcCode	Gets the register value (EIIC or FEIC).
OSIllegalException_SystemRegister_ExcPC	Gets the register value (EIPC or FEPC).
OSErrorGetServiceId	Gets the system service identifier.
OSError_SystemService_Parameter	Gets the parameters.

Remark The AUTOSAR specifications do not prescribe utility functions InitApplicationInterrupts, _kernel_fv0_InitializeIntService, OSIllegalException_SystemRegister_ExcCode and OSIllegalException_SystemRegister_ExcPC.
These are our original utility functions added to the RV850.

InitApplicationInterrupts

[Overview]

Enables acceptance of interrupts.

[Issue scope]

Tasks

[Syntax]

```
void InitApplicationInterrupts ( void );
```

[Parameters]

None

[Function]

Allows acceptance of interrupts to the OS-Application to which the processing program that issued this utility function belongs.

Remark 1. This utility function can be issued only when scalability class 3 (SC3) is defined for the Scalability class "OsScalabilityClass".

Remark 2. This utility function manipulates the EI level interrupt mask register (IMR*m*) in the interrupt acceptance enable processing.
The IMR*m* corresponding to the interrupt service routine registered in the target OS-Application is manipulated.

Remark 3. It is assumed that this utility function is issued from a task (Task identifier "OsRestartTask") that shifts from SUSPENDED state to READY state when TerminateApplication is issued with parameter *RestartOption* set to RESTART(0x1).

Remark 4. If this utility function is issued from a processing program outside the valid scope of issue, it will not be handled as an error and no processing will be performed.

Remark 5. When an OS-Application is terminated or restarted, the RV850 does not clear the interrupts whose acceptance is suspended.
Therefore, if these suspended interrupts need to be cleared when the OS-Application is restarted, the user should execute the interrupt clear processing before issuing this utility function.

[Return values]

None

_kernel_fv0_InitializeIntService

[Overview]

Enables the issuing of system services for interrupt handling.

[Issue scope]

Boot process

[Syntax]

```
void _kernel_fv0_InitializeIntService ( void );
```

[Parameters]

None

[Function]

Executes the initialization processing to enable the issuing of the System Services for Interrupt Handling provided by the RV850 before StartOS is issued.

Remark 1. This utility function can be issued only before StartOS is issued.

Remark 2. Since this utility function manipulates the RV850 internal data, it must be issued in supervisor mode (the UM bit in PSW is 0).

[Return values]

None

OSIllegalException_SystemRegister_ExcCode

[Overview]

Gets the register value (EIIC or FEIC).

[Issue scope]

Common hook routine (ShutdownHook), OS-Application-specific hook routine (ShutdownHook_*OsApplication*)

[Syntax]

```
SystemRegisterType OSIllegalException_SystemRegister_ExcCode ( void );
```

[Parameters]

None

[Function]

Gets the register value (EIIC or FEIC), and returns the value.

Remark This utility function can be issued only before the processing program issues any system service.

[Return values]

Macro	Numerical Value	Description
-	0x0 - 0xFFFFFFFF	System register value

OSIllegalException_SystemRegister_ExcPC

[Overview]

Gets the register value (EIPC or FEPC).

[Issue scope]

Common hook routine (ShutdownHook), OS-Application-specific hook routine (ShutdownHook_*OsApplication*)

[Syntax]

```
SystemRegisterType OSIllegalException_SystemRegister_ExcPC ( void );
```

[Parameters]

None

[Function]

Gets the register value (EIPC or FEPC), and returns the value.

Remark This utility function can be issued only before the processing program issues any system service.

[Return values]

Macro	Numerical Value	Description
-	0x0 - 0xFFFFFFFF	System register value

OSErrorGetServiceId

[Overview]

Gets the system service identifier.

[Issue scope]

Common hook routine (ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
OSServiceIdType OSErrorGetServiceId ( void );
```

[Parameters]

None

[Function]

Gets the identifier of the target system service (the system service that caused activation of the processing program that issued this utility function), and returns the system service identifier.

Remark 1. This utility function can be issued only before the processing program issues any system service.

Remark 2. When the processing program (ErrorHook or ErrorHook_*OsApplication*) that issues this utility function was called for a reason other than "abnormal end of the system service", this utility function will return an undefined value.

[Return values]

Macro	Numerical Value	Description
OSServiceID_GetApplicationID	0x0	System service identifier of GetApplicationID
OSServiceID_GetISRID	0x1	System service identifier of GetISRID
OSServiceID_CallTrustedFunction	0x2	System service identifier of CallTrustedFunction
OSServiceId_CheckISRMemoryAccess	0x3	System service identifier of CheckISRMemoryAccess
OSServiceId_CheckTaskMemoryAccess	0x4	System service identifier of CheckTaskMemoryAccess
OSServiceId_CheckObjectAccess	0x5	System service identifier of CheckObjectAccess
OSServiceId_CheckObjectOwnership	0x6	System service identifier of CheckObjectOwnership
OSServiceId_StartScheduleTableRel	0x7	System service identifier of StartScheduleTableRel
OSServiceId_StartScheduleTableAbs	0x8	System service identifier of StartScheduleTableAbs
OSServiceId_StopScheduleTable	0x9	System service identifier of StopScheduleTable
OSServiceId_NextScheduleTable	0xA	System service identifier of NextScheduleTable

Macro	Numerical Value	Description
OSServiceId_GetScheduleTableStatus	0xE	System service identifier of GetScheduleTableStatus
OSServiceId_IncrementCounter	0xF	System service identifier of IncrementCounter
OSServiceId_GetCounterValue	0x10	System service identifier of GetCounterValue
OSServiceId_GetElapsedValue	0x11	System service identifier of GetElapsedValue
OSServiceId_TerminateApplication	0x12	System service identifier of TerminateApplication
OSServiceID_AllowAccess	0x13	System service identifier of AllowAccess
OSServiceID_GetApplicationState	0x14	System service identifier of GetApplicationState
OSServiceId_StartOS	0x40	System service identifier of StartOS
OSServiceId_ShutdownOS	0x41	System service identifier of ShutdownOS
OSServiceId_GetActiveApplicationMode	0x42	System service identifier of GetActiveApplicationMode
OSServiceId_ActivateTask	0x43	System service identifier of ActivateTask
OSServiceId_TerminateTask	0x44	System service identifier of TerminateTask
OSServiceId_ChainTask	0x45	System service identifier of ChainTask
OSServiceId_Schedule	0x46	System service identifier of Schedule
OSServiceId_GetTaskID	0x47	System service identifier of GetTaskID
OSServiceId_GetTaskState	0x48	System service identifier of GetTaskState
OSServiceId_EnableAllInterrupts	0x49	System service identifier of EnableAllInterrupts
OSServiceId_DisableAllInterrupts	0x4A	System service identifier of DisableAllInterrupts
OSServiceId_ResumeAllInterrupts	0x4B	System service identifier of ResumeAllInterrupts
OSServiceId_SuspendAllInterrupts	0x4C	System service identifier of SuspendAllInterrupts
OSServiceId_ResumeOSInterrupts	0x4D	System service identifier of ResumeOSInterrupts
OSServiceId_SuspendOSInterrupts	0x4E	System service identifier of SuspendOSInterrupts
OSServiceId_GetResource	0x4F	System service identifier of GetResource
OSServiceId_ReleaseResource	0x50	System service identifier of ReleaseResource
OSServiceId_SetEvent	0x51	System service identifier of SetEvent
OSServiceId_ClearEvent	0x52	System service identifier of ClearEvent
OSServiceId_GetEvent	0x53	System service identifier of GetEvent
OSServiceId_WaitEvent	0x54	System service identifier of WaitEvent
OSServiceId_GetAlarmBase	0x55	System service identifier of GetAlarmBase
OSServiceId_GetAlarm	0x56	System service identifier of GetAlarm
OSServiceId_SetRelAlarm	0x57	System service identifier of SetRelAlarm
OSServiceId_SetAbsAlarm	0x58	System service identifier of SetAbsAlarm
OSServiceId_CancelAlarm	0x59	System service identifier of CancelAlarm

OSError_*SystemService_Parameter*

[Overview]

Gets the parameters.

[Issue scope]

Common hook routine (ErrorHook), OS-Application-specific hook routine (ErrorHook_*OsApplication*)

[Syntax]

```
TaskType OSError_ActivateTask_TaskID ( void );
TaskType OSError_ChainTask_TaskID ( void );
TaskRefType OSError_GetTaskID_TaskID ( void );
TaskType OSError_GetTaskState_TaskID ( void );
TaskStateRefType OSError_GetTaskState_State ( void );
ResourceType OSError_GetResource_ResID ( void );
ResourceType OSError_ReleaseResource_ResID ( void );
TaskType OSError_SetEvent_TaskID ( void );
EventMaskType OSError_SetEvent_Mask ( void );
EventMaskType OSError_ClearEvent_Mask ( void );
TaskType OSError_GetEvent_TaskID ( void );
EventMaskRefType OSError_GetEvent_Event ( void );
EventMaskType OSError_WaitEvent_Mask ( void );
CounterType OSError_IncrementCounter_CounterID ( void );
CounterType OSError_GetCounterValue_CounterID ( void );
TickRefType OSError_GetCounterValue_Value ( void );
CounterType OSError_GetElapsedValue_CounterID ( void );
TickRefType OSError_GetElapsedValue_Value ( void );
TickRefType OSError_GetElapsedValue_ElapsedValue ( void );
AlarmType OSError_GetAlarmBase_AlarmID ( void );
AlarmBaseRefType OSError_GetAlarmBase_Info ( void );
AlarmType OSError_GetAlarm_AlarmID ( void );
TickRefType OSError_GetAlarm_Tick ( void );
AlarmType OSError_SetRelAlarm_AlarmID ( void );
TickType OSError_SetRelAlarm_increment ( void );
TickType OSError_SetRelAlarm_cycle ( void );
AlarmType OSError_SetAbsAlarm_AlarmID ( void );
```

`TickType OSError_SetAbsAlarm_start ( void );`
`TickType OSError_SetAbsAlarm_cycle ( void );`
`AlarmType OSError_CancelAlarm_AlarmID ( void );`
`ScheduleTableType OSError_StartScheduleTableRel_ScheduleTableID ( void );`
`TickType OSError_StartScheduleTableRel_Offset ( void );`
`ScheduleTableType OSError_StartScheduleTableAbs_ScheduleTableID ( void );`
`TickType OSError_StartScheduleTableAbs_Start ( void );`
`ScheduleTableType OSError_StopScheduleTable_ScheduleTableID ( void );`
`ScheduleTableType OSError_NextScheduleTable_ScheduleTableID_From ( void );`
`ScheduleTableType OSError_NextScheduleTable_ScheduleTableID_To ( void );`
`ScheduleTableType OSError_GetScheduleTableStatus_ScheduleTableID ( void );`
`ScheduleTableStatusRefType OSError_GetScheduleTableStatus_ScheduleStatus ( void );`
`TrustedFunctionIndexType OSError_CallTrustedFunction_FunctionIndex ( void );`
`TrustedFunctionParameterRefType OSError_CallTrustedFunction_FunctionParams ( void );`
`ApplicationType OSError_TerminateApplication_Application ( void );`
`RestartType OSError_TerminateApplication_RestartOption ( void );`
`ApplicationType OSError_GetApplicationState_Application ( void );`
`ApplicationStateRefType OSError_GetApplicationState_Value ( void );`

Remark 1. Specify a system service name for *SystemService* in this utility function name and a parameter for *Parameter*.

Remark 2. Only a *StatusType*-type system service that has a parameter can be specified for *SystemService* in this utility function name.

[Parameters]

None

[Function]

Gets the value or the pointer address specified for the parameter in the target system service (the system service that caused activation of the processing program that issued this utility function), and returns the value or pointer address.

Remark When the processing program (ErrorHook or ErrorHook_*OsApplication*) that issues this utility function was called for a reason other than "abnormal end of the system service", this utility function will return an undefined value.

[Return values]

For details of the return value, see [Parameters] for each system service in "14.4System Services Reference".

A. CONFIGURATOR

This appendix describes the configurator.

A.1 Overview

The configurator is a utility tool for reading a CF file as the input file and outputting information files (SIT files, ENTRY files, and kernel macro files).

A.2 Activation Method

The following describes the method for starting the configurator from the command prompt.
Here, "C:\" represents the command prompt, the "Δ" character represents input of the Space key, and text surrounded by square brackets ("[ ]") represents startup options that can be omitted.

```
C:\> Os_Configurator.exe Δ [cf_file] Δ [@cmd_file] Δ [-o Δ sit_file] Δ [-no] Δ [-e Δ entry_file] Δ [-ne] Δ [-nk] Δ [-I Δ path_name] Δ [-nsc] Δ [-V] Δ [-help]
```

Each activation option is described in detail below.

(1) *cf_file*
Treats the CF file (ARXML format or OIL format) specified by *cf_file* as the input file.
Note that the maximum length of the *cf_file* parameter is 255 characters.

Remark 1. If the *cf_file* string includes ASCII space characters, such as "Program Files", the *cf_file* parameter must be surrounded by double quotation marks.

Remark 2. In the configurator, the *cf_file* is handled as an ARXML format when the extention is .arxml or .xml, as an OIL format when the extention is .oil.
Only .arxml, .xml or .oil can be specified as extension of *cf_file*.

Remark 3. *cf_file* can be specified multiple times as shown below. However, only the last parameter specified is treated as a valid CF file, and the others will be treated as invalid files. Consequently, in the example below, cf_file3.oil is a valid CF file, and the other files (cf_file1.arxml and cf_file2.xml) are ignored.

```
C:\> Os_Configurator.exe Δ cf_file1.arxml Δ cf_file2.xml Δ cf_file3.oil
```

Remark 4. When *cf_file* is in the ARXML format, the configurator assumes that the descriptions (specified items) in the file match the AUTOSAR_RENESAS_OS_ECUConfigurationParameters.arxml contents (the specified items are equivalent to those in the OIL format).

Remark 5. The AUTOSAR specifications do not prescribe that an OIL-format file conforming to the specifications of OSEK Implementation Language Version 2.5 can be specified as an input file. In the RV850, this type of file can be specified as an input file.

[If omitted:]
The configurator searches the Os_Configurator.exe startup folder to find if the following files exist, and if one of these files is found, it is treated as the CF file.

- Os_Config.arxml (Priority in search: First)
- Os_Config.xml (Priority in search: Second)
- Os_Config.oil (Priority in search: Third)

(2) *@cmd_file*
Treats the file specified by *cmd_file* as a command file.
Note that the maximum length of the *cmd_file* parameter is 255 characters.

Remark If the *cmd_file* string includes ASCII space characters, such as "Program Files", the *cmd_file* parameter must be surrounded by double quotation marks.

(3) -o Δ *sit_file*
Outputs the SIT file with the file name specified by *sit_file*.
Note that the maximum length of the *sit_file* parameter is 255 characters.

Remark 1. If the *sit_file* string includes ASCII space characters, such as "Program Files", the *sit_file* parameter must be surrounded by double quotation marks.

Remark 2. -o Δ *sit_file* can be specified multiple times as shown below. However, only the first -o Δ *sit_file* specified is treated as a valid activation option, and the others will be treated as invalid activation options.
Consequently, in the example below, -o Δ sit_file1.c is a valid activation option, and the other activation options (-o Δ sit_file2.c and -o Δ sit_file3.c) are ignored.

```
C:\> Os_Configurator.exe Δ -o Δ sit_file1.c Δ -o Δ sit_file2.c Δ -o Δ sit_file3.c Δ cf_file.oil
```

[If omitted:]
The process is executed assuming that "-o Δ Os_Cfg.c" was specified.

(4) -no
Outputs no SIT file.

(5) -e Δ *entry_file*
Outputs the ENTRY file with the file name specified by *entry_file*.
Note that the maximum length of the *entry_file* parameter is 255 characters.

Remark 1. If the *entry_file* string includes ASCII space characters, such as "Program Files", the *entry_file* parameter must be surrounded by double quotation marks.

Remark 2. -e Δ *entry_file* can be specified multiple times as shown below. However, only the first -e Δ *entry_file* specified is treated as a valid activation option, and the others will be treated as invalid activation options.
Consequently, in the example below, -e Δ entry_file1.850 is a valid activation option, and the other activation options (-e Δ entry_file2.850 and -e Δ entry_file3.850) are ignored.

```
C:\> Os_Configurator.exe Δ -e Δ entry_file1.850 Δ -e Δ entry_file2.850 Δ -e Δ entry_file3.850 Δ cf_file.oil
```

[If omitted:]
The process is executed assuming that "-e Δ Os_CfgEntry.850" was specified.

(6) -ne
Outputs no ENTRY file.

(7) -nk
Outputs no kernel macro file.

[If omitted:]
Outputs the kernel macro file "Os_Cfg.h".

(8) -I Δ *path_name*
Specifies the folder to search for the Include Files written in the CF file (OIL).
Note that the maximum length of the *path_name* parameter is 255 characters.

Remark 1. If the *path_name* string includes ASCII space characters, such as "Program Files", the *path_name* parameter must be surrounded by double quotation marks.

Remark 2. This activation option can be specified multiple times (up to 255 times) as shown below.

```
C:\> Os_Configurator.exe Δ -I Δ path_name1 Δ -I Δ path_name2 Δ -I Δ path_name3 Δ cf_file.oil
```

Remark 3. The search order for include files differs as shown below, depending on the format of the CF file (OIL) definition.

["#include Δ <*inc_file*>" format]

- Folder specified by *path_name*
- Folder storing *cf_file*

- Startup folder of Os_Configurator.exe

["#include Δ "*inc_file*"" format]

- Folder storing *cf_file*
- Startup folder of Os_Configurator.exe
- Folder specified by *path_name*

[If omitted:]
Include files are searched for in the following order.

- Folder storing *cf_file*
- Startup folder of Os_Configurator.exe

(9) -nsc
Treats the C++ style comments ("//") in the CF file (OIL) as errors.

[If omitted:]
The C++ style comments ("//") in the CF file (OIL) are not treated as errors.

(10) -V
Outputs the version information of the configurator.

Remark When this activation option is specified, all other activation options are disabled and output of the information file is suppressed.

[If omitted:]
Does not output the version information.

(11) -help
Outputs help messages (information about types, usage, etc.) for the activation options of the configurator.

Remark When this activation option is specified, all other activation options are disabled and output of the information file is suppressed.

[If omitted:]
Does not output help messages for the activation options.

A.2.1 Command file

In the configurator, use of a command file is supported to eliminate the limitation on the maximum length of characters that can be specified in the activation options in the command line.
The format of the command file is described below.

(1) Comment
All text after a hash symbol (#) until the end of the line is treated as a comment.
Comments can be written in SJIS or EUC encoding.

(2) Activation option separator
Activation options can be separated by a line break, an ASCII space, or a tab.
When an activation option consists of a command part (*-xxx*) and a parameter part, such as "-o Δ *sit_file*", "-e Δ *entry_file*", or "-I Δ *path_name*", the command (*-xxx*) and parameter can be separated by a line break, an ASCII space, or a tab.

(3) Character strings storing ASCII spaces
If an activation option (such as "*cf_file*", "*cmd_file*", "-o Δ *sit_file*", "-e Δ *entry_file*", or "-I Δ *path_name*") stores an ASCII space (e.g. "Program Files"), it must be surrounded by double quotation marks.

Figure A.1 Coding Example of Command File

```
# Command file
C:\sample\src\cf_file.oil           # CF file
-o C:\sample\src\sit_file.c         # SIT file
-e C:\sample\src\entry_file.850     # ENTRY file
-I "C:\Program Files\sample\inc"    # Include path
```

A.3 Sample Command Input

The following shows examples of command input for the configurator.
Here, "C:\>" represents the command prompt, and the "Δ" character represents input of the Space key.

(1) After the CF file cf_file.oil is read from the folder C:\Program Files\sample, the information files (SIT file: Os_Cfg.c; ENTRY file: Os_CfgEntry.850; and kernel macro file: Os_Cfg.h) are output to the folder where Os_Configurator.exe was started.
The include files defined in cf_file.oil are searched for in the following order.

- Folder storing cf_file.oil
- Startup folder of Os_Configurator.exe

```
C:\> Os_Configurator.exe Δ "C:\Program Files\sample\cf_file.oil"
```

(2) The file cf_file.oil stored in the folder specified by the path ..\sample relative to the startup folder of Os_Configurator.exe is read as the input file, and then the information file (SIT file: sit_file.c) is output to the folder specified by the path ..\sample relative to the startup folder of Os_Configurator.exe.
The include files defined in cf_file.oil are searched for in the following order.

- Folder storing cf_file.oil
- Startup folder of Os_Configurator.exe

```
C:\> Os_Configurator.exe Δ ..\sample\cf_file.oil Δ -o Δ ..\sample\sit_file.c Δ -
ne Δ -nk
```

(3) After the file cf_file.oil is read as the CF file from the startup folder of Os_Configurator.exe, the information files (SIT file: Os_Cfg.c; ENTRY file: Os_CfgEntry.850; and kernel macro file: Os_Cfg.h) are output to the current folder where Os_Configurator.exe was started.
The include files defined in cf_file.oil are searched for in the following order.

["#include Δ <*inc_file*>" format]

- The folder specified by the path ..\sample\inc relative to the startup folder of Os_Configurator.exe
- Folder storing cf_file.oil
- Startup folder of Os_Configurator.exe

["#include Δ "*inc_file*"" format]

- Folder storing cf_file.oil
- Startup folder of Os_Configurator.exe
- The folder specified by the path ..\sample\inc relative to the startup folder of Os_Configurator.exe

```
C:\> Os_Configurator.exe Δ cf_file.oil Δ -I Δ ..\sample\inc
```

(4) The version information of the configurator is output.

```
C:\> Os_Configurator.exe Δ -V
```

A.4 Messages

Messages are generated and output if information of which the user should be notified (e.g. invalid activation options and specification of out-of-bounds numbers) is detected while the configurator is executing a process.

There are three types of message, according to the importance of the notification.

- E: Fatal error
  If a specified number of fatal errors occur, the configurator aborts processing. No information file is output.
- F: Abort error
  The configurator aborts processing. No information file is output.
- W: Warning
  The configurator continues processing. Information files are output.
  The contents of the output information files may differ from those expected by the user.

A.4.1 Fatal errors

The following shows the messages that are output when fatal errors are detected.
In each message, the italic text indicates what is determined when the corresponding fatal error is detected.

Table A.1 Fatal Errors

E2001	[Message]	Nesting of include is greater than 8.
	[Description]	Nesting of include files exceeds 8 levels.
	[Action by user]	Reduce the level of include-file nesting to no more than 8, for example by combining multiple include files into one file.
E2002	[Message]	Object name is longer than 255 characters.
	[Description]	The object name exceeds the maximum length of 255 characters.
	[Action by user]	Change the name of the object so that it is no more than 255 characters.
E2003	[Message]	Syntax error.
	[Description]	There is a syntax error.
	[Action by user]	Check for any syntax errors.
E2005	[Message]	*information* is expected.
	[Description]	Required information *information* is missing.
	[Action by user]	Add the description of information *information*.
E2006	[Message]	Illegal description (*string*).
	[Description]	Invalid string *string* is specified.
	[Action by user]	Specify a valid string.
E2007	[Message]	Out of range (*value*).
	[Description]	Specification of numerical value *value* is invalid.
	[Action by user]	Specify a numerical value within the valid range.
E2008	[Message]	*xxx* sub part is expected when *yyy* is *zzz*.
	[Description]	When *zzz* is specified for *yyy*, *xxx* must not be omitted.
	[Action by user]	Add *xxx*.

E2009	[Message]	*xxx* sub part is unnecessary when *yyy* is *zzz*.
	[Description]	When *zzz* is specified for *yyy*, *xxx* must not be specified.
	[Action by user]	Delete *xxx*.
E2010	[Message]	There is no required item in *xxx*.
	[Description]	The definition of a required item in *xxx* is missing.
	[Action by user]	Add the item.
E3000	[Message]	Identifier is conflict (*name*).
	[Description]	Name *name* is declared multiple times.
	[Action by user]	Change one of the names.
E3001	[Message]	*xxx* is multiple defined (*yyy*).
	[Description]	*xxx* is defined multiple times in *yyy*.
	[Action by user]	Avoid multiple definitions of *xxx*.
E3002	[Message]	Too many objects (*xxx*).
	[Description]	The object *xxx* is defined more than the maximum allowable number of times.
	[Action by user]	Define the object *xxx* no more than the maximum allowable number of times.
E3005	[Message]	*information* is not aligned.
	[Description]	Information *information* is not aligned.
	[Action by user]	Specify a value that is aligned to the prescribed value.
E3007	[Message]	RV850 doesn't support a value larger than 32 bit size.
	[Description]	A value greater than 0xFFFFFFFF is specified.
	[Action by user]	Specify 0xFFFFFFFF or a smaller value.
E3102	[Message]	*xxx* is not defined in *yyy*.
	[Description]	Information *xxx* corresponding to identifier *yyy* is not defined.
	[Action by user]	Define information *xxx*.
E3103	[Message]	*xxx* can not defined in *yyy*.
	[Description]	*xxx* cannot be defined in *yyy*.
	[Action by user]	Delete *xxx*.
E3104	[Message]	Timing Protection is defined for Category 1 ISR (*xxx*).
	[Description]	A definition related to timing protection has been made for a category 1 interrupt service routine.
	[Action by user]	Change the category of the interrupt service routine from 1 to 2, or delete the definition of the timing protection.
E3105	[Message]	Setting in a memory area is the illegal value.
	[Description]	The specified range of the memory area is invalid.
	[Action by user]	Specify valid addresses.

E3106	[Message]	The size and the termination address can't be specified at the same time.
	[Description]	Both the size and end address are specified.
	[Action by user]	Use either the size or the end address to specify a memory area.
E3107	[Message]	*xxx* is lower than *yyy*.
	[Description]	A value smaller than *yyy* is specified for *xxx*.
	[Action by user]	Specify *yyy* or a greater value for *xxx*.
E3108	[Message]	*xxx* is higher than *yyy*.
	[Description]	A value greater than *yyy* is specified for *xxx*.
	[Action by user]	Specify *yyy* or a smaller value for *xxx*.
E3109	[Message]	There are a lot of specification of *xxx* (*yyy*).
	[Description]	The total number of *xxx* definitions in *yyy* is too many.
	[Action by user]	Reduce the number of *xxx* definitions.
E3110	[Message]	Internal resource is multiple defined (*xxx*).
	[Description]	The internal resource is defined multiple times.
	[Action by user]	Reduce the internal resource definitions to one.
E3112	[Message]	*xxx* is OS reserved.
	[Description]	*xxx* is a reserved word and cannot be used.
	[Action by user]	Change the name without using a reserved word.
E3113	[Message]	*xxx* doesn't belong to yyy.
	[Description]	Object *xxx* does not belong to OS-Application *yyy*.
	[Action by user]	Make object *xxx* belong to OS-Application *yyy*.
E4002	[Message]	*xxx* must be *yyy* for *zzz*.
	[Description]	*yyy* must be defined in *xxx* with the condition *zzz*.
	[Action by user]	Specify an appropriate value for *xxx* according to the *zzz* condition.
E4003	[Message]	*xxx* cannot be defined by *yyy*.
	[Description]	*xxx* cannot be defined with the condition *yyy*.
	[Action by user]	To define *xxx*, condition *yyy* should be checked and modified.
E4006	[Message]	No one use this *object_type* (*object_name*).
	[Description]	Object *object_name* with object type *object_type* is not used anywhere.
	[Action by user]	Delete the object information corresponding to *object_name*.
E4007	[Message]	Not enabled to access (*xxx*) (*yyy*).
	[Description]	*xxx* does not have access privileges to *yyy*.
	[Action by user]	Give *xxx* access privileges to *yyy*.
E4008	[Message]	Invalid event name.
	[Description]	An event that is not assigned to a task is specified.
	[Action by user]	Check and modify the event identifier.

E4009	[Message]	Category 1 ISR's priority must be set to a higher priority than Category 2 ISR's priority.
	[Description]	The priority of category 1 interrupt service routines is lower than that of category 2 interrupt service routines.
	[Action by user]	Define a higher priority for category 1 interrupt service routines than that of category 2 interrupt service routines.
E4010	[Message]	There is no space in event mask (*xxx*).
	[Description]	AUTO cannot be specified because there is no unused event bit.
	[Action by user]	Specify a value from 0x1 to 0xFFFFFFFF as the event mask.
E4011	[Message]	TASK or ISR is expected in OS-Application (*xxx*).
	[Description]	There are no tasks or interrupt service routines in the OS-Application information.
	[Action by user]	Make at least one task or interrupt service routine belong to the OS-Application information.
E4013	[Message]	*object_type* (*object_name*) doesn't belong to application.
	[Description]	Object *object_name* with object type *object_type* does not belong to any OS-Application.
	[Action by user]	Make object *object_name* belong to an OS-Application.
E4014	[Message]	Cyclic chain of alarm actions with Increment Counter (*xxx*).
	[Description]	Counter *xxx* to be manipulated repeats a loop operation.
	[Action by user]	Check the validity of the associated counter and the counter to be manipulated in expiry action.
E4015	[Message]	*object_name* is defined in multiple OS-Application.
	[Description]	The same object *object_name* is defined in multiple OS-Application information entries.
	[Action by user]	Check and modify the identifier.
E4017	[Message]	The offset value of action is the same value.
	[Description]	The same expiry count value is defined in another location.
	[Action by user]	Change one of the expiry count values.
E4019	[Message]	The same exception code as other ISR is specified.
	[Description]	The same exception code is defined in another location.
	[Action by user]	Change one of the exception codes.
E4020	[Message]	When *xxx* option is used, *yyy* should be defined.
	[Description]	When *xxx* is specified for activation option, *yyy* must not be omitted.
	[Action by user]	Add *yyy*, or don't specify activation option *xxx*.

A.4.2 Abort errors

The following shows the messages that are output when abort errors are detected.
In each message, the italic text indicates what is determined when the corresponding abort error is detected.

Table A.2 Abort Errors

F1000	[Message]	Can not open file (*file_name*).
	[Description]	Cannot open file *file_name*.
	[Action by user]	Check whether the file exists.
F1002	[Message]	Unknown device file format.
	[Description]	The format of the device file is invalid.
	[Action by user]	Reinstall the device file.
F1003	[Message]	Too long file name.
	[Description]	The file name exceeds the maximum length of 255 characters.
	[Action by user]	Change the name of the file so that it is no more than 255 characters.
F1004	[Message]	Too long folder name.
	[Description]	The folder name exceeds the maximum length of 255 characters.
	[Action by user]	Change the name of the folder so that it is no more than 255 characters.
F1005	[Message]	Output file names are the same (*file_name*).
	[Description]	Output file names *file_name* are conflicting.
	[Action by user]	Avoid conflict between the SIT file name specified by activation option -o and the ENTRY file name specified by activation option -e.
F1006	[Message]	Unauthorized use of option (*option*).
	[Description]	Illegal specification format of activation option *option*.
	[Action by user]	Check the specification format of the activation option.
F1007	[Message]	Illegal option (*option*).
	[Description]	Illegal specification of activation option *option*.
	[Action by user]	Check the specification format of the activation option.
F1008	[Message]	Illegal format in command file.
	[Description]	Illegal specification format of the command file.
	[Action by user]	Check the specification format of the command file.
F1009	[Message]	Out of memory.
	[Description]	Not enough memory.
	[Action by user]	Release memory (e.g. by exiting resident programs).
F1010	[Message]	option *option* is expected.
	[Description]	Option *option* is a mandatory activation option.
	[Action by user]	Specify activation option *option*.

F1011	[Message]	Too many include path.
	[Description]	More than 255 include paths are defined.
	[Action by user]	Reduce the number of include path definitions to no more than 255, for example by combining multiple include paths.
F1012	[Message]	Too many File lines.
	[Description]	The number of lines in the CF file (OIL) exceeds 32767.
	[Action by user]	Use default assumptions, etc. to reduce the number of lines in the CF file (OIL) to no more than 32767.
F1013	[Message]	Configuration file is not specified.
	[Description]	No CF file specified.
	[Action by user]	Specify a CF file.
F6002	[Message]	XML file syntax error.
	[Description]	An XML syntax error has occurred.
	[Action by user]	Check the syntax in the CF file (XML).
F6003	[Message]	Unknown error.
	[Description]	An unknown error has occurred.
	[Action by user]	Check the descriptions in the CF file (XML).
F6004	[Message]	XML file syntax error by an AUTOSAR schema.
	[Description]	A syntax error has occurred due to the AUTOSAR schema.
	[Action by user]	Check the syntax in the CF file (XML).

A.4.3 Warnings

The following shows the messages that are output when warnings are detected.
In each message, the italic text indicates what is determined when the corresponding warning is detected.

Table A.3 Warnings

W1000	[Message]	Nested command file.
	[Description]	The command file is nested. Processing is continued with command-file nesting ignored.
	[Action by user]	Calling a command file from another command file is prohibited. Include the contents of the called another command file in the calling command file.
W4001	[Message]	Priority is lower than *process*'s priority. (Rise to value of process's priory)
	[Description]	The specified ceiling value is illegal. Processing is continued with the assumption that the same value as the priority of processing program *process* is defined.
	[Action by user]	Change the ceiling value to a higher priority than that of the processing program that uses the resource.

Remark The AUTOSAR specifications do not prescribe a warning equivalent to W4001. This is our original warning item added to the RV850.

B. CF FILES (OIL)

This appendix describes how to write a CF file (OIL) in conformance with the OSEK Implementation Language Version 2.5 specification.

B.1 Overview

A CF file (OIL) is required to generate files that hold the Configuration Information to be supplied to the RV850 (SIT files, ENTRY files, and kernel macro files). These files are written by the user with a text editor.

The following describes the notation used in a CF file (OIL).

(1) Character encoding
A CF file (OIL) is written in ASCII code.
Comments can be written in SJIS or EUC encoding.

(2) Comments
Text surrounded by "/*" and "*/" and the rest of lines following a "//" are treated as comments.

(3) Numerical values
Words starting with a number from 0 to 9 are treated as numerical values.
Numbers storing a decimal point must be written in decimal notation.

Words starting with numbers 0 to 9:	Decimal
Words starting with 0x or 0X:	Hexadecimal

Remark Numbers in a CF file (OIL) cannot be written in binary or octal notation.

(4) Names
Words starting with an alphabetic character (a to z or A to Z) or an underscore "_" are treated as names.
Up to 255 characters can be specified as a name.

Remark 1. Names and symbol names are distinguished according to the context of the CF file (OIL).

Remark 2. The CF file (OIL) is case-sensitive for alphabetic characters (a to z and A to Z are handled as different characters).

(5) Symbol names
Words starting with an alphabetic character (a to z or A to Z) or an underscore "_" are treated as symbol names.
Up to 4095 characters can be specified as a symbol name.

Remark 1. Names and symbol names are distinguished according to the context of the CF file (OIL).

Remark 2. The CF file (OIL) is case-sensitive for alphabetic characters (a to z and A to Z are handled as different characters).

Remark 3. In the RV850, symbol names starting with _kernel or _KERNEL are OS reserved symbols and must not be used for any other purpose than the specified one.

(6) Keywords
The words shown below are reserved as keywords for CF files (OIL) and must not be used for any other purpose than the specified one.

ABSOLUTE, ACCESSING_APPLICATION, ACTION, ACTIVATETASK, ACTIVATION, ADJUSTABLEEXPPOINT, ALARM, ALARMCALLBACK, ALARMCALLBACKNAME, ALARMTIME, APPLICATION, APPMODE, ATTRIBUTE, AUTO, AUTOSTART, CATEGORY, CODE, CONST, CONSTNAME, CORE, CORENUMBER, COUNTER, CPU, CPUCORE, CYCLETIME, DATA, DEFAULTFPSRVALUE, DRIVER, ECUCPARTITION, ENDADDRESS, ERRORHOOK, EVENT, EXCEPTIONCODE, EXECUTIONBUDGET, EXPLICIT, EXTENDED, FALSE, FULL, G3K, G3M, G3KH, G3MH, HARDWARE, IMPLICIT, INCREMENTCOUNTER, INTC1EICTRL, INTC1EIMASK, INTC2EICTRL, INTC2EIMASK, INTERNAL, INTERRUPTBASE, INTPRI0 to INTPRI15, ISR, IOC, LENGTH, LINKED, LINKEDRESOURCE, LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION, LOCKINGTIME, MASK, MAXADVANCE, MAXALLINTERRUPTLOCKTIME, MAXALLOWEDVALUE, MAXEXCEPTIONCODE, MAXOSINTERRUPTLOCKTIME, MAXRESOURCELOCKTIME, MAXRETARD, MEMORYAREA, MINCYCLE, NAME, NON, NONE, OFFSET, OS, OSDEFAULTAPPMODE, OSTMCNT, PRECISION, PERIODIC, POSTTASKHOOK, PRETASKHOOK, PRIORITY, PROTECT, PROTECTIONHOOK, READONLY, READWRITE, RELATIVE, RESOURCE, RESOURCEPROPERTY, RESOURCELOCK, RES_SCHEDULER, RESTARTTASK, SAVEFPUREG, SC1 to SC4, SCALABILITYCLASS, SCHEDULE, SCHEDULETABLE, SECONDSPERTICK, SETEVENT, SHUTDOWNHOOK, SIZE, SOFTWARE, SPID, SPINLOCK, STANDARD, STACKMONITORING, STACKSIZE, STARTADDRESS, STARTUPHOOK, STARTVALUE, STATUS, SYN-

CHRON, SYNCSTRATEGY, SYSTEM, SYSTEM_CLOCK, TASK, TICKSPERBASE, TIMECONSTANTS, TIMEFRAME, TIMEVALUE, TIMING_PROTECTION, TRACESYSTEMENTRY, TRACESYSTEMEXIT, TRACETASKSTATUS, TRUE, TRUSTED, TRUSTED_FUNCTION, TYPE, USEGETSERVICEID, USEPARAMETERACCESS, USERESSCHEDULER

B.2 Configuration Information

The data (configuration information) described in a CF file (OIL) is broadly classified into the following types.

(1) Include Files

(2) CPU

(a) Alarm information

(b) Application mode information

(c) OS-Application information

(d) Counter information

(e) Event information

(f) Interrupt service routine information

(g) OS information

(h) Resource information

(i) Schedule table information

(j) Task information

(k) System information

The following shows a schematic image of a CF file (OIL). Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.1 CF File (OIL) Coding Sample

```
// Include Files
[ #include inc_file ]

// CPU
CPU Os {
    [ Alarm information ]
    [ Application mode information ]
    [ OS-Application information ]
    [ Counter information ]
    [ Event information ]
    [ Interrupt service routine information ]
    OS information
    [ Resource information ]
    [ Schedule table information ]
    [ Task information ]
    System information
};
```

B.3 Include Files

The following item is defined as required information for other CF files (OIL).

- Include file "inc_file"

The format for defining an include file is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.2 Include File Definition Format

```
#include inc_file
```

(1) Include file "*inc_file*"
Specifies the name of an include file.
Only a name (up to 255 characters including a path) can be specified as *inc_file*.

Remark 1. The search order for include files differs as shown below, depending on the format of the *inc_file* definition.

[In "#include <*inc_file*>" format]

- Folder specified by *path_name* in activation option "-I Δ *path_name*"
- Folder storing the CF file (OIL) specified by activation option "*cf_file*"
- Startup folder of configurator Os_Configurator.exe

[In "#include "*inc_file*"" format]

- Folder storing the CF file (OIL) specified by activation option "*cf_file*"
- Startup folder of configurator Os_Configurator.exe
- Folder specified by *path_name* in activation option "-I Δ *path_name*"

Remark 2. The maximum nesting level of include files is eight.

Remark 3. This item can be defined multiple times as shown below.

```
#include "inc_file1.oil"
#include "inc_file2.oil"
#include "inc_file3.oil"
```

B.4 CPU

The following items are defined as required information for implementing the functions provided by the RV850.

- Identifier "Os"
- Alarm information to system information

Only one CPU can be defined.
The format for defining the CPU is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.3 CPU Definition Format

```
CPU Os {
      [ Alarm information ]
      [ Application mode information ]
      [ OS-Application information ]
      [ Counter information ]
      [ Event information ]
      [ Interrupt service routine information ]
      OS information
      [ Resource information ]
      [ Schedule table information ]
      [ Task information ]
      System information
};
```

(1) Identifier "*Os*"
Specifies the CPU identifier.
Only a name can be specified as *Os*.

(2) Alarm information to system information
See "B.4.1Alarm information" to "B.4.11System information" for details about Alarm information to System information.

B.4.1 Alarm information

The following items are defined as required information for implementing the ALARM MANAGEMENT provided by the RV850.

- Identifier "OsAlarm"
- OS-Application identifier "OsAlarmAccessingApplication"
- Counter identifier "OsAlarmCounterRef"
- Expiry action "OsAlarmAction"
  - Task identifier "OsAlarmActivateTaskRef"
  - Event identifier "OsAlarmSetEventRef"
  - Task identifier "OsAlarmSetEventTaskRef"
  - Counter identifier "OsAlarmIncrementCounterRef"
  - Alarm callback identifier "OsAlarmCallbackName"
- Initial state "OsAlarmAutostart"
  - Expiry count value "OsAlarmAlarmTime"
  - Count mode "OsAlarmAutostartType"
  - Cycle count value "OsAlarmCycleTime"
  - Application mode "OsAlarmAppModeRef"

The sum of the numbers of defined alarm information sets and Schedule table information sets must be between 0 and 1023.

Remark In the RV850, if SC3 is defined for Scalability class "OsScalabilityClass" and the counter defined in this information is not defined in the OS-Application information, fatal error E4013 will be output.

The format for defining alarm information is shown below. Strings surrounded by square brackets "[ ]" are optional items.

Figure B.4 Alarm Information Definition Format

```
ALARM OsAlarm {
    [ ACCESSING_APPLICATION = OsAlarmAccessingApplication; ]
    COUNTER = OsAlarmCounterRef;
    ACTION = OsAlarmAction {
        [ TASK = OsAlarmActivateTaskRef; ]
        [ EVENT = OsAlarmSetEventRef; ]
        [ TASK = OsAlarmSetEventTaskRef; ]
        [ COUNTER = OsAlarmIncrementCounterRef; ]
        [ ALARMCALLBACKNAME = OsAlarmCallbackName; ]
    };
    [ AUTOSTART = OsAlarmAutostart {
        [ ALARMTIME = OsAlarmAlarmTime; ]
        [ TYPE = OsAlarmAutostartType; ]
        [ CYCLETIME = OsAlarmCycleTime; ]
        [ APPMODE = OsAlarmAppModeRef; ]
    }; ]
};
```

(1) Identifier "*OsAlarm*"
Specifies the alarm identifier.
Only a name can be specified as *OsAlarm*.

(2) OS-Application identifier "*OsAlarmAccessingApplication*"
Specifies the identifier of an OS-Application that defines objects (tasks, interrupt service routines, and counters) to which access privileges to this alarm should be assigned.
Only Identifier "OsApplication" can be specified as *OsAlarmAccessingApplication*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".

The AUTOSAR specifications prescribe that a warning should be output when a value other than SC3 and SC4 is defined for Scalability class "OsScalabilityClass". In the RV850, however, fatal error E4003 will be output.

Remark 2. It is not necessary to specify the identifier of the OS-Application to which this alarm belongs as *OsAlarmAccessingApplication*.

Remark 3. This item can be specified multiple times (up to 31 times) as shown below.

```
ACCESSING_APPLICATION = OsApplication1;
ACCESSING_APPLICATION = OsApplication2;
ACCESSING_APPLICATION = OsApplication3;
```

[If omitted:]
Processing is performed assuming that access privileges to this alarm are assigned only to the objects (tasks, interrupt service routines, and counters) defined in the OS-Application to which this alarm belongs.

(3) Counter identifier "*OsAlarmCounterRef*"
Specifies the identifier of the counter with which this alarm is to be associated (the counter that holds the count for confirming whether the expiry conditions have been met).
Only Identifier "OsCounter" can be specified as *OsAlarmCounterRef*.

Remark When SC3 is defined for Scalability class "OsScalabilityClass" and "a counter that does not belong to the OS-Application to which this alarm belongs (counter belonging to another OS-Application)" is specified for this counter identifier, the identifier of the OS-Application to which the counter belongs should be defined for OS-Application identifier "OsAlarmAccessingApplication".

(4) Expiry action "*OsAlarmAction*"
Specifies the type of processing to perform when the expiry conditions are met.
Only ACTIVATETASK, SETEVENT, INCREMENTCOUNTER, or ALARMCALLBACK can be specified as *OsAlarmAction*.

ACTIVATETASK:	Task activation (processing equivalent to ActivateTask)
SETEVENT:	Setting of event mask (processing equivalent to SetEvent)
INCREMENTCOUNTER:	Update of count value (processing equivalent to IncrementCounter)
ALARMCALLBACK:	Activation of alarm callback (only in SC1)

Remark ALARMCALLBACK can be specified only when SC1 is defined for Scalability class "OsScalabilityClass".

(a) Task identifier "*OsAlarmActivateTaskRef*"
Specifies the identifier of the task to activate when the expiry conditions are met.
Only Identifier "OsTask" can be specified as *OsAlarmActivateTaskRef*.

Remark 1. This item can be specified only when ACTIVATETASK is defined for Expiry action "OsAlarmAction".

Remark 2. When SC3 is defined for Scalability class "OsScalabilityClass" and "a task that does not belong to the OS-Application to which this alarm belongs (task belonging to another OS-Application)" is specified for this task identifier, the identifier of the OS-Application to which the alarm belongs as OS-Application identifier "OsTaskAccessingApplication".

(b) Event identifier "*OsAlarmSetEventRef*"
Specifies the identifier of the event that holds the event mask to set when the expiry conditions are met.
Only Identifier "OsEvent" can be specified as *OsAlarmSetEventRef*.

Remark 1. This item can be specified only when SETEVENT is defined for Expiry action "OsAlarmAction".

Remark 2. The event specified here must be assigned to the task defined for Event identifier "OsAlarmSetEventRef".

(c) Task identifier "*OsAlarmSetEventTaskRef*"
Specifies the identifier of the task in which the event mask is to be set when the expiry conditions are met.
Only Identifier "OsTask" can be specified as *OsAlarmSetEventTaskRef*.

Remark 1. This item can be specified only when SETEVENT is defined for Expiry action "OsAlarmAction".

Remark 2. When SC3 is defined for Scalability class "OsScalabilityClass" and "a task that does not belong to the OS-Application to which this alarm belongs (task belonging to another OS-Application)" is specified for this task identifier, the identifier of the OS-Application to which the alarm belongs as OS-Application identifier "OsTaskAccessingApplication".

(d) Counter identifier "*OsAlarmIncrementCounterRef*"
Specifies the identifier of the counter whose count value is to be updated when the expiry conditions are met.
Only Identifier "OsCounter" can be specified as *OsAlarmIncrementCounterRef*.

Remark 1. This item can only be specified when INCREMENTCOUNTER is defined for Expiry action "OsAlarmAction".

Remark 2. This item can specify only the counters with Type "OsCounterType" defined as SOFTWARE.

Remark 3. When SC3 is defined for Scalability class "OsScalabilityClass" and "a counter that does not belong to the OS-Application to which this alarm belongs (counter belonging to another OS-Application)" is specified for this item, the identifier of the OS-Application to which this alarm belongs as OS-Application identifier "OsTaskAccessingApplication".

Remark 4. The AUTOSAR specifications prescribe that when this item and Counter identifier "OsAlarmCounterRef" are set to the same value (loop operation), a warning will be output. In the RV850, however, fatal error E4014 will be output.

(e) Alarm callback identifier "*OsAlarmCallbackName*"
Specifies the identifier of the alarm callback to be called when the expiry conditions are met.
Only a symbol name (up to 4095 characters) can be specified as *OsAlarmCallbackName*.

Remark 1. This item can be specified only when ALARMCALLBACK is defined for Expiry action "OsAlarmAction".

Remark 2. When an alarm callback is defined as follows, the value set in *OsAlarmCallbackName* should be "OsAlarmCallbackName1". See "8.2Alarm Callback" for details about alarm callback.

```
ALARMCALLBACK ( OsAlarmCallbackName1 ) {
    ..................
    ..................
    return;
}
```

(5) Initial state "*OsAlarmAutostart*"
Specifies the initial state of the alarm.
Only TRUE or FALSE can be specified as *OsAlarmAutostart*.

TRUE: Depends on StartOS parameter "*Mode*"
FALSE: Inactive state

Remark 1. When this item is set to "TRUE", the initial state of the alarm changes as follows according to the value set in StartOS parameter "*Mode*".

[Values of "*Mode*" and Application mode "OsAlarmAppModeRef" match]

- Active state

[Values of "*Mode*" and Application mode "OsAlarmAppModeRef" do not match]

- Inactive state

Remark 2. The format for specifying FALSE is as follows:

```
AUTOSTART = FALSE;
```

Remark 3. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

(a) Expiry count value "*OsAlarmAlarmTime*"
Specifies the expiry count value of the alarm (relative count value or absolute count value).
The value that can be specified for *OsAlarmAlarmTime* depends on the definition of Type "OsCounterType" of the counter associated through Counter identifier "OsAlarmCounterRef".

- ABSOLUTE
  Only a value from 0x0 to Maximum count value "OsCounterMaxAllowedValue" can be specified.

- RELATIVE
  Only a value from 0x1 to Maximum count value "OsCounterMaxAllowedValue" can be specified.

Remark Only a relative count value can be specified when RELATIVE is defined for Count mode "OsAlarmAutostartType" or an absolute count value when ABSOLUTE is defined.

(b) Count mode "*OsAlarmAutostartType*"
Specifies the mode for counting until Expiry count value "OsAlarmAlarmTime". Only RELATIVE or ABSOLUTE can be specified as *OsAlarmAutostartType*.

RELATIVE: Relative count
ABSOLUTE: Absolute count

(c) Cycle count value "*OsAlarmCycleTime*"
Specifies the cycle count value of the alarm.
Only 0x0 or a value from Minimum cycle value "OsCounterMinCycle" to Maximum count value "OsCounterMaxAllowedValue" can be specified as *OsAlarmCycleTime*.

Remark When 0x0 is specified in *OsAlarmCycleTime*, the alarm operates as a one-shot alarm. When a value other than 0x0 is specified, it operates as a cycle alarm.

(d) Application mode "*OsAlarmAppModeRef*"
Specifies the application mode of the alarm.
Only Application mode "OsAppMode" can be specified for *OsAlarmAppModeRef*.

Remark 1. Processing is performed assuming that the default application mode OSDEFAULTAPPMODE is defined regardless of whether it is actually defined.

Remark 2. This item can be specified multiple times (up to 127 times) as shown below.

```
APPMODE = OSDEFAULTAPPMODE;
APPMODE = OsAppMode1;
APPMODE = OsAPPMode2;
```

[If omitted:]
The AUTOSAR specifications prescribe that this item must not be omitted. In the RV850, however, when this item is omitted, processing is performed assuming that OSDEFAULTAPPMODE is specified.

B.4.2 Application mode information

The following items are defined as required information for implementing the OS EXECUTION MANAGEMENT provided by the RV850.

- Application mode "OsAppMode"

0 to 127 sets of application mode information can be specified.
The format for defining application mode information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.5 Application Mode Information Definition Format

```
APPMODE OsAppMode { };
```

(1) Application mode "*OsAppMode*"
Specifies the application mode.
Only a name or OSDEFAULTAPPMODE can be specified as *OsAppMode*.

Name:	Normal application mode
OSDEFAULTAPPMODE:	Default application mode

Remark Processing is performed assuming that the default application mode OSDEFAULTAPPMODE is defined, regardless of whether it is actually defined.

[If omitted:]
The AUTOSAR specifications prescribe that this item must not be omitted. In the RV850, however, when this item is omitted, processing is performed assuming that the following is defined.

```
APPMODE OSDEFAULTAPPMODE { };
```

B.4.3 OS-Application information

The following items are defined as required information for implementing the OS-APPLICATION MANAGEMENT provided by the RV850.

- Identifier "OsApplication"
- Reliability "OsTrusted"
- OS-Application Stack size "OsAppStackSize"
- FPSR "OsAppDefaultFPSRValue"
- Core ID "OsApplicationCoreAssignment"
- SPID "OsApplicationSPID"
- Alarm identifier "OsAppAlarmRef"
- Counter identifier "OsAppCounterRef"
- Interrupt service routine identifier "OsAppIsrRef"
- Schedule table identifier "OsAppScheduleTableRef"
- Task identifier "OsAppTaskRef"
- Task identifier "OsRestartTask"
- StartupHook_OsApplication "OsAppStartupHook"
- ShutdownHook_OsApplication "OsAppShutdownHook"
- ErrorHook_OsApplication "OsAppErrorHook"
- Trusted function "OsApplicationTrustedFunction"
    - Identifier "OsTrustedFunctionName"
- Memory area identifier "OsAppMemoryAreaNameRef"
    - Attribute "OsAppMemoryAreaAttribute"

0 to 31 sets of OS-Application information can be defined.

Remark The AUTOSAR specifications prescribe the item ECUCPARTITION as OS-Application information. This item is not supported, however, on the RV850.

The format for defining OS-Application information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.6 OS-Application Information Definition Format

```
APPLICATION OsApplication {
    TRUSTED = OsTrusted;
    STACKSIZE = OsAppStackSize;
    [ DEFAULTFPSRVALUE = OsAppDefaultFPSRValue; ]
    [ CORE = OsApplicationCoreAssignment; ]
    [ SPID = OsApplicationSPID; ]
    [ ALARM = OsAppAlarmRef; ]
    [ COUNTER = OsAppCounterRef; ]
    [ ISR = OsAppIsrRef; ]
    [ SCHEDULETABLE = OsAppScheduleTableRef; ]
    [ TASK = OsAppTaskRef; ]
    [ RESTARTTASK = OsRestartTask; ]
    STARTUPHOOK = OsAppStartupHook;
    SHUTDOWNHOOK = OsAppShutdownHook;
    ERRORHOOK = OsAppErrorHook;
    [ TRUSTED_FUNCTION = OsApplicationTrustedFunction {
        NAME = OsTrustedFunctionName;
    }; ]
    [ MEMORYAREA = OsAppMemoryAreaNameRef {
        ATTRIBUTE = OsAppMemoryAreaAttribute;
```

```
    }; ]
};
```

(1) Identifier "*OsApplication*"
Specifies the OS-Application identifier
Only a name can be specified as *OsApplication*.

Remark This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass". The AUTOSAR specifications prescribe that a warning should be output when a value other than SC3 and SC4 is defined for Scalability class "OsScalabilityClass". In the RV850, however, fatal error E4003 will be output.

(2) Reliability "*OsTrusted*"
Specifies the reliability of the OS-Application.
Only TRUE or FALSE can be specified as *OsTrusted*.

TRUE: Trusted OS-Application
FALSE: Non-trusted OS-Application

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(3) OS-Application Stack size "*OsAppStackSize*"
Specifies the size (in bytes) of the stack used by this OS-Application.
Only a 0x4-byte aligned value from 0x4 to 0x0FFFFFFC can be specified for *OsAppStackSize*.

Remark 1. See "C.7.2OS-Application stack" for details about the size to be specified in this item.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(4) FPSR "*OsAppDefaultFPSRValue*"
Specifies the value to be set in the floating-point configuration/status register (FPSR) when a processing program (a task, an interrupt service routine, or a hook routine) belonging to this OS-Application is activated.
Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsAppDefaultFPSRValue*.

Remark 1. This item can be specified only when TRUE is defined for FPSR saving/restoring "OsSaveFpuReg".

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

[If omitted:]
Processing is performed assuming that the same value as FPSR default value "OsDefaultFPSRValue" is specified.

(5) Core ID "*OsApplicationCoreAssignment*"
Specifies the ID of the target core to be controlled by the RV850.
Only 0x1 can be specified as *OsApplicationCoreAssignment*.

Remark The AUTOSAR specifications prescribe that 0 to 65534 can be specified for this item. In the RV850, however, only 0x1 can be specified for this item.

[If omitted:]
Processing is performed assuming that 0x1 is specified.

(6) SPID "*OsApplicationSPID*"
Specifies the access privileges (system protection identifier) to the I/O area.
Only a value from 0x0 to 0x3 can be specified as *OsApplicationSPID*.

Remark 1. This item can be specified when G3M or G3KH or G3MH is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted".

Remark 2. When TRUE is defined for Reliability "OsTrusted" or when 0x0 is specified for this item, any access to the I/O area is enabled.

Remark 3. The correspondence between the value specified in *OsAppicationSPID* and access rights to the I/O area depends on the Boot Process operation extracted as user own coding modules.

Remark 4. See the user's manual of the target device for details on the SPID.

Remark 5. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(7) Alarm identifier "*OsAppAlarmRef*"
Specifies the identifier of the alarm to be assigned to the OS-Application.
Only Identifier "OsAlarm" can be specified as *OsAppAlarmRef*.

Remark This item can be specified multiple times (up to 1023 times) as shown below.

```
ALARM = OsAlarm1;
ALARM = OsAlarm2;
ALARM = OsAlarm3;
```

(8) Counter identifier "*OsAppCounterRef*"
Specifies the identifier of the counter to be assigned to the OS-Application.
Only Identifier "OsCounter" can be specified as *OsAppCounterRef*.

Remark This item can be specified multiple times (up to 1023 times) as shown below.

```
COUNTER = OsCounter1;
COUNTER = OsCounter2;
COUNTER = OsCounter3;
```

(9) Interrupt service routine identifier "*OsAppIsrRef*"
Specifies the identifier of the interrupt service routine to be assigned to the OS-Application.
Only Identifier "OsIsr" can be specified as *OsAppIsrRef*.

Remark 1. A category 1 interrupt service routine can be defined only for a trusted OS-Application (TRUE is defined for Reliability "OsTrusted").

Remark 2. This item can be specified multiple times (up to 1023 times) as shown below.

```
ISR = OsIsr1;
ISR = OsIsr2;
ISR = OsIsr3;
```

Remark 3. The AUTOSAR specifications prescribe that both Task identifier "OsAppTaskRef" and this item can be left undefined. However, if neither is defined in the RV850, fatal error E4011 will be output.

(10) Schedule table identifier "*OsAppScheduleTableRef*"
Specifies the identifier of the schedule table to be assigned to the OS-Application.
Only Identifier "OsScheduleTable" can be specified as *OsAppScheduleTableRef*.

Remark This item can be specified multiple times (up to 1023 times) as shown below.

```
SCHEDULETABLE = OsScheduleTable1;
SCHEDULETABLE = OsScheduleTable2;
SCHEDULETABLE = OsScheduleTable3;
```

(11) Task identifier "*OsAppTaskRef*"
Specifies the identifier of the task to be assigned to the OS-Application.
Only Identifier "OsTask" can be specified as *OsAppTaskRef*.

Remark 1. This item can be specified multiple times (up to 1023 times) as shown below.

```
TASK = OsTask1;
TASK = OsTask2;
TASK = OsTask3;
```

Remark 2. The AUTOSAR specifications prescribe that both Interrupt service routine identifier "OsAppIsrRef" and this item can be left undefined. In the RV850, however, if neither is defined, fatal error E4011 will be output.

(12) Task identifier "*OsRestartTask*"
Specifies the identifier of the task that issues an activation request when TerminateApplication is issued (with parameter *RestartOption* set to RESTART).
Only Task identifier "OsAppTaskRef" can be specified for *OsRestartTask*.

[If omitted:]
Processing is performed assuming that parameter *RestartOption* is not set to RESTART when TerminateApplication issued.

(13) StartupHook_*OsApplication* "*OsAppStartupHook*"
Specifies whether to assign OS-Application-specific hook routine StartupHook_*OsApplication* to this OS-Application.
Only TRUE or FALSE can be specified as *OsAppStartupHook*.

TRUE: The OS-Application-specific hook routine is assigned.
FALSE: No OS-Application-specific hook routine is assigned.

Remark 1. When TRUE is specified, the following OS-Application-specific hook routine is assigned to this OS-Application.

```
void
StartupHook_OsApplication ( void ) {
    ..................
    ..................
}
```

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(14) ShutdownHook_*OsApplication* "*OsAppShutdownHook*"
Specifies whether to assign OS-Application-specific hook routine ShutdownHook_*OsApplication* to this OS-Application.
Only TRUE or FALSE can be specified as *OsAppShutdownHook*.

TRUE: The OS-Application-specific hook routine is assigned.
FALSE: No OS-Application-specific hook routine is assigned.

Remark 1. When TRUE is specified, the following OS-Application-specific hook routine is assigned to this OS-Application.

```
void
ShutdownHook_OsApplication ( StatusType Fatalerror ) {
..................
..................
}
```

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(15) ErrorHook_*OsApplication* "*OsAppErrorHook*"
Specifies whether to assign OS-Application-specific hook routine ErrorHook_*OsApplication* to this OS-Application.
Only TRUE or FALSE can be specified as *OsAppErrorHook*.

TRUE: The OS-Application-specific hook routine is assigned.
FALSE: No OS-Application-specific hook routine is assigned.

Remark 1. When TRUE is specified, the following OS-Application-specific hook routine is assigned to this OS-Application.

```
void
ErrorHook_OsApplication ( StatusType Error ) {
..................
..................
}
```

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(16) Trusted function "*OsApplicationTrustedFunction*"
Specifies whether to assign a trusted function to this OS-Application.
Only TRUE or FALSE can be specified as *OsApplicationTrustedFunction*.

TRUE: A trusted function is assigned.
FALSE: No trusted function is assigned.

Remark 1. This item can be specified only when TRUE is defined for Reliability "OsTrusted".

Remark 2. This item can be specified multiple times (up to 1023 times) as shown below.

```
TRUSTED_FUNCTION = TRUE {
    NAME = OsTrustedFunctionName1;
};
TRUSTED_FUNCTION = TRUE {
    NAME = OsTrustedFunctionName2;
};
TRUSTED_FUNCTION = TRUE {
    NAME = OsTrustedFunctionName3;
};
```

Remark 3. The format for specifying FALSE is as follows:

```
TRUSTED_FUNCTION = FALSE;
```

Remark 4. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

(a) Identifier "*OsTrustedFunctionName*"
Specifies the identifier of the trusted function.
Only a name can be specified as *OsTrustedFunctionName*.

Remark 1. When the trusted function is defined as follows, the value set in *OsTrustedFunctionName* should be "OsTrustedFunctionName1".

```
TRUSTED ( OsTrustedFunctionName1 ) {
    ..................
    ..................
}
```

Remark 2. The AUTOSAR specifications prescribe that only a function name can be specified for this item. In the RV850, however, only an identifier can be specified for this item.

(17) Memory area identifier "*OsAppMemoryAreaNameRef*"
Specifies the identifier of a memory area as the target of access protection in this OS-Application.
Only Memory area identifier "OsSystemMemoryArea" can be specified for *OsAppMemoryAreaNameRef*.

Remark 1. This item can be specified when G3M or G3KH or G3MH is defined for Core identifier "OsSystem-CpuCore", and FALSE is defined for Reliability "OsTrusted".

Remark 2. This item can be specified multiple times (up to 4 times) as shown below.

```
MEMORYAREA = OsSystemMemoryArea11 {
    ATTRIBUTE = CODE;
};
MEMORYAREA = OsSystemMemoryArea12 {
    ATTRIBUTE = CONST;
};
MEMORYAREA = OsSystemMemoryArea13 {
    ATTRIBUTE = DATA;
};
```

Remark 3. The memory area to be protected in common in all OS-Applications should be defined for Memory area identifier "OsMemoryAreaNameRef".

Remark 4. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(a) Attribute "*OsAppMemoryAreaAttribute*"
Specifies the attribute to be assigned to the memory area.
Only CODE, CONST, or DATA can be specified as *OsAppMemoryAreaAttribute*.

CODE: Readable and executable
CONST: Readable
DATA: Readable and writable

Remark 1. If a processing program (a task, an interrupt service routine, or a hook routine) belonging to this OS-Application performs memory access other than that specified in this item, processing is performed (ProtectionHook is called or ShutdownOS is issued) in accordance with the definition in ProtectionHook "OsProtectionHook".

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

B.4.4 Counter information

The following items are defined as required information for implementing the COUNTER MANAGEMENT provided by the RV850.

- Identifier "OsCounter"
- Maximum count value "OsCounterMaxAllowedValue"
- Minimum cycle value "OsCounterMinCycle"
- Basic count value "OsCounterTicksPerBase"
- Type "OsCounterType"
- Number of seconds per tick "OsSecondsPerTick"
- OS-Application identifier "OsCounterAccessingApplication"
- Hardware counter information
  - Exception code "OsCounterExceptionCode"
  - Priority "OsCounterPriority"
- Data macro information
  - Macro name "OsConstName"
  - Count value "OsTimeValue"

0 to 1023 sets of counter information can be defined.

Remark 1. In the RV850, when the counter defined in this information is not used in Alarm information or Schedule table information, fatal error E4006 will be output.

Remark 2. In the RV850, if SC3 is defined for Scalability class "OsScalabilityClass" and the counter defined in this information is not defined in the OS-Application information, fatal error E4013 will be output.

The format for defining counter information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.7 Counter Information Definition Format

```
COUNTER OsCounter {
    MAXALLOWEDVALUE = OsCounterMaxAllowedValue;
    MINCYCLE = OsCounterMinCycle;
    TICKSPERBASE = OsCounterTicksPerBase;
    TYPE = OsCounterType;
    [ SECONDSPERTICK = OsSecondsPerTick; ]
    [ ACCESSING_APPLICATION = OsCounterAccessingApplication; ]
    [ DRIVER {
        EXCEPTIONCODE = OsCounterExceptionCode;
        [ PRIORITY = OsCounterPriority; ]
    }; ]
    [ TIMECONSTANTS {
        CONSTNAME = OsConstName;
        TIMEVALUE = OsTimeValue;
    }; ]
};
```

(1) Identifier "*OsCounter*"
Specifies the counter identifier.
Only a name or SYS_COUNTER can be specified as *OsCounter*.

Name:	Normal counter
SYS_COUNTER:	System counter

Remark See "7.1.1System counters" for details about the system counter.

(2) Maximum count value "*OsCounterMaxAllowedValue*"
Specifies the maximum value that can be counted using the counter. Only a value from Minimum cycle value "OsCounterMinCycle" to 0x7FFFFFFF can be specified as *OsCounterMaxAllowedValue*.

Remark The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFFFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 0x7FFFFFFF.

(3) Minimum cycle value "*OsCounterMinCycle*"
Specifies the minimum cycle value that can be defined for cycle processing (alarms and schedule tables) using the counter. Only a value from 0x1 to Maximum count value "OsCounterMaxAllowedValue" can be specified for *OsCounterMinCycle*.

Remark The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFFFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 0x7FFFFFFF.

(4) Basic count value "*OsCounterTicksPerBase*"
Specifies the basic count value of the counter.
Only a value from 0x1 to 0x7FFFFFFF can be specified as *OsCounterTicksPerBase*.

Remark The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 0x7FFFFFFF.

(5) Type "*OsCounterType*"
Specifies the counter type.
Only SOFTWARE or HARDWARE can be specified as *OsCounterType*.

SOFTWARE: Software counter
HARDWARE: Hardware counter

Remark See "7.1Overview" for details about the software counter and hardware counter.

(6) Number of seconds per tick "*OsSecondsPerTick*"
Specifies the number of seconds per tick (in seconds).
Only a value from 0.000001 to 1.000000 can be specified as *OsSecondsPerTick*.

Remark 1. This item can be specified only when HARDWARE is defined for Type "OsCounterType".

Remark 2. The AUTOSAR specifications prescribe that a value from 0 to INF can be specified for this item. In the RV850, however, a value from 0.000001 to 1.000000 can be specified for this item.

(7) OS-Application identifier "*OsCounterAccessingApplication*"
Specifies the identifier of an OS-Application that defines objects (tasks, interrupt service routines, and alarms) to which access privileges to this counter should be assigned.
Only Identifier "OsApplication" can be specified as *OsCounterAccessingApplication*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass". The AUTOSAR specifications prescribe that a warning should be output when a value other than SC3 and SC4 is defined for Scalability class "OsScalabilityClass". In the RV850, however, fatal error E4003 will be output.

Remark 2. It is not necessary to specify the identifier of the OS-Application to which this counter belongs as *OsCounterAccessingApplication*.

Remark 3. This item can be specified multiple times (up to 31 times) as shown below.

```
ACCESSING_APPLICATION = OsApplication1;
ACCESSING_APPLICATION = OsApplication2;
ACCESSING_APPLICATION = OsApplication3;
```

[If omitted:]
Processing is performed assuming that access privileges to this counter are assigned only to the objects (tasks, interrupt service routines, and alarms) defined in the OS-Application to which the counter belongs.

(8) Hardware counter information
Defines information regarding the interrupt service routine (category 2) that performs update processing of the hardware counter.

Remark This item can be specified only when HARDWARE is defined for Type "OsCounterType".

(a) Exception code "*OsCounterExceptionCode*"
Specifies the exception code for the EI level interrupt causing activation of the interrupt service routine.
Only a value from 0x1000 to Maximum exception code "OsSystemMaxExceptionCode" can be specified for *OsCounterExceptionCode*.

Remark The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(b) Priority "*OsCounterPriority*"
Specifies the priority of the interrupt defined for Exception code "OsCounterExceptionCode". Only a value from NTPRI0 to INTPRI15 (or INTPRI0 to INTPRI7 when the target device is G3K) can be specified for *OsCounterPriority*.

Remark 1. For the value defined in *OsCounterPriority*, INTPRI0 signifies the lowest priority and INTPRI15 signifies the highest priority.

Remark 2. *OsCounterPriority* must not be set to a priority equal to or higher than Initial priority "OsIsrPriority" of the interrupt service routine for which 0x1 is specified as Category "OsIsrCategory".

Remark 3. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

[If omitted:]
Processing is performed assuming that INTPRI0 is specified.

(9) Data macro information
This should be defined only when the results of converting a length of time from units of seconds to units of ticks (value calculated by Count value "OsTimeValue" / Number of seconds per tick "OsSecondsPerTick") are output to an information file (kernel macro file) as a data macro.

Remark 1. This item can be specified only when HARDWARE is defined for Type "OsCounterType".

Remark 2. This item can be specified only when Number of seconds per tick "OsSecondsPerTick" is defined.

Remark 3. This item can be specified multiple times (up to 1023 times) as shown below.

```
TIMECONSTANTS {
    CONSTNAME = OSConstMs;
    TIMEVALUE = 0.1;
};
TIMECONSTANTS {
    CONSTNAME = OSConstSec;
    TIMEVALUE = 1;
};
TIMECONSTANTS {
    CONSTNAME = OSConstMin;
    TIMEVALUE = 60;
};
```

(a) Macro name "*OsConstName*"
Specifies the macro name to be used when a processing program references data.
Only a name can be specified as *OsConstName*.

(b) Count value "*OsTimeValue*"
Specifies the value (in seconds) for converting a length of time from units of seconds to units of ticks.
Only a value from 0.000001 to 999.999999 can be specified as *OsTimeValue*.

Remark The AUTOSAR specifications prescribe that a value from 0 to INF can be specified for this item.
In the RV850, however, a value from 0.000001 to 999.999999 can be specified for this item.

B.4.5 Event information

The following items are defined as required information for implementing the EVENT MANAGEMENT provided by the RV850.

- Identifier "OsEvent"
- Event mask "OsEventMask"

0 to 1023 sets of event information can be defined.
The format for defining event information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Remark In the RV850, if the event defined in this information is not defined in the Task information, fatal error E4006 will be output.

Figure B.8 Event Information Definition Format

```
EVENT OsEvent {
[ MASK = OsEventMask; ]
};
```

(1) Identifier "*OsEvent*"
Specifies the event identifier.
Only a name can be specified as *OsEvent*.

(2) Event mask "*OsEventMask*"
Specifies the event mask.
Only a value from 0x1 to 0xFFFFFFFF or AUTO can be specified as *OsEventMask*.

Remark 1. When AUTO is specified, the lowest-order unused event bit of all tasks to which this event is assigned is found, and the value obtained by setting the found unused bit to "1" is assigned as the event mask.
Consequently, if a definition is made as shown below, the third bit is the lowest-order unused event bit, so the OsEvent3 event mask will be "0x4".

```
TASK OsTask1 {
    ..................
    ..................
    EVENT = OsEvent1;
    EVENT = OsEvent3;
    ..................
    ..................
};
TASK OsTask2 {
    ..................
    ..................
    EVENT = OsEvent2;
    EVENT = OsEvent3;
    ..................
    ..................

};
EVENT OsEvent1 {
    MASK = 0x3;
};
EVENT OsEvent2 {
    MASK= 0x8;
};
EVENT OsEvent3 {
    MASK = AUTO;
};
```

Remark 2. The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFFFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 0xFFFFFFFF.

Remark 3. In the ARXML format, specify 0 to select the processing equivalent to AUTO.

[If omitted:]
Processing is performed assuming that AUTO is specified.

B.4.6 Interrupt service routine information

The following items are defined as required information for implementing the INTERRUPT HANDLING provided by the RV850.

- Identifier "OsIsr"
- Category "OsIsrCategory"
- Exception code "OsIsrExceptionCode"
- Initial priority "OsIsrPriority"
- Resource identifier "OsIsrResourceRef"
- Timing protection "OsIsrTimingProtection"

0 to 1023 sets of interrupt service routine information can be defined.

Remark 1. The AUTOSAR specifications do not prescribe the operation when neither the Task information nor this information is defined. In the RV850, fatal error E4011 will be output in this case.

Remark 2. In the RV850, if SC3 is defined for Scalability class "OsScalabilityClass" and the interrupt service routine defined in this information is not defined in the OS-Application information, fatal error E4013 will be output.

The format for defining interrupt service routine information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.9 Interrupt Service Routine Information Definition Format

```
ISR OsIsr {
    CATEGORY = OsIsrCategory;
    EXCEPTIONCODE = OsIsrExceptionCode;
    [ PRIORITY = OsIsrPriority; ]
    [ RESOURCE = OsIsrResourceRef; ]
    [ TIMING_PROTECTION = OsIsrTimingProtection; ]
};
```

(1) Identifier "*OsIsr*"
Specifies the interrupt service routine identifier. Only a name can be specified as *OsIsr*.

Remark When the interrupt service routine is defined as follows, the value set in *OsIsr* should be "OsIsr1".

```
ISR ( OsIsr1 ) {
    ...................
    ...................
}
```

(2) Category "*OsIsrCategory*"
Specifies the category of the interrupt service routine.
Only 0x1 or 0x2 can be specified as *OsIsrCategory*.

0x1: Category 1
0x2: Category 2

Remark When SC3 is defined for Scalability class "OsScalabilityClass", 0x1 can be specified only when TRUE is defined for Reliability "OsTrusted".

(3) Exception code "*OsIsrExceptionCode*"
Specifies the exception code for the EI level interrupt causing activation of the interrupt service routine.
Only a value from 0x1000 to Maximum exception code "OsSystemMaxExceptionCode" can be specified for *OsIsrExceptionCode*.

Remark 1. This item can be specified multiple times (up to 512 times) as shown below.

```
EXCEPTIONCODE = 0x1000;
EXCEPTIONCODE = 0x1010;
EXCEPTIONCODE = 0x1020;
```

Remark 2. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(4) Initial priority "*OsIsrPriority*"
Specifies the initial priority of the interrupt defined for Exception code "OsIsrExceptionCode".
The value that can be specified for *OsIsrPriority* depends on the target device type; INTPRI0 to INTPRI7 when the target device is G3K or INTPRI0 to INTPRI15 for other devices.

Remark 1. For the value defined in *OsIsrPriority*, INTPRI0 signifies the lowest priority and INTPRI15 signifies the highest priority.

Remark 2. *OsIsrPriority* of the category 1 interrupt service routine should be set to a priority higher than that of any category 2 interrupt service routine (including Priority "OsCounterPriority" specified for hardware counters).

Remark 3. When 0x1 is defined for Category "OsIsrCategory", this item cannot be set to the same value as Priority "OsCounterPriority".

Remark 4. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

[If omitted:]
Processing is performed assuming that INTPRI0 is specified.

(5) Resource identifier "*OsIsrResourceRef*"
Specifies the identifier of the resource that is manipulated (acquired or released) by the interrupt service routine.
Only Identifier "OsResource" can be specified as *OsIsrResourceRef*.

Remark 1. This item can be specified only when 0x2 is defined for Category "OsIsrCategory".

Remark 2. It is not possible to specify an "internal resource identifier" for this item.

Remark 3. When SC3 is defined for Scalability class "OsScalabilityClass" and "a resource that does not belong to the OS-Application to which this interrupt service routine belongs (resource belonging to another OS-Application)" is specified for this resource identifier, the identifier of the OS-Application to which the interrupt service routine belongs as OS-Application identifier "OsResourceAccessingApplication".

Remark 4. This item can be specified multiple times (up to 1023 times) as shown below.

```
RESOURCE = OsResource1;
RESOURCE = OsResource2;
RESOURCE = OsResource3;
```

[If omitted:]
This interrupt service routine is processed assuming that it does not manipulate (acquire or release) resources.

(6) Timing protection "*OsIsrTimingProtection*"
Specifies whether to use the timing protection function when interrupt service routine processing is performed.
Only FALSE can be specified as *OsIsrTimingProtection*.

FALSE: Not used

Remark 1. The AUTOSAR specifications prescribe that use of the timing protection function should be disabled when SC1 or SC3 is defined for Scalability class "OsScalabilityClass".
In the RV850, when an illegal value is specified for this item, fatal error E4003 will be output.

Remark 2. In the ARXML format, specify 0 for this item to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

B.4.7 OS information

The following items are defined as the basic information required for RV850 operation.

- Identifier "OsOS"
- Maximum number of cores "OsNumberOfCores"
- Scalability class "OsScalabilityClass"
- Stack monitoring facilities "OsStackMonitoring"
- Status type "OsStatus"
- OSErrorGetServiceId "OsUseGetServiceId"
- OSError_SystemService_Parameter "OsUseParameterAccess"
- Scheduler resource "OsUseResScheduler"
- FPSR saving/restoring "OsSaveFpuReg"
- Base address "OsInterruptBaseAddress"
- System stack size "OsStackSize"
- FPSR default value "OsDefaultFPSRValue"
- StartupHook "OsStartupHook"
- ShutdownHook "OsShutdownHook"
- PostTaskHook "OsPostTaskHook"
- PreTaskHook "OsPreTaskHook"
- ErrorHook "OsErrorHook"
- ProtectionHook "OsProtectionHook"
- Memory area identifier "OsMemoryAreaNameRef"
    - Attribute "OsMemoryAreaAttribute"

Only one set of OS information can be defined.
The format for defining OS information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.10 OS Information Definition Format

```
OS OsOS {
    [ CORENUMBER = OsNumberOfCores; ]
    [ SCALABILITYCLASS = OsScalabilityClass; ]
    STACKMONITORING = OsStackMonitoring;
    STATUS = OsStatus;
    USEGETSERVICEID = OsUseGetServiceId;
    USEPARAMETERACCESS = OsUseParameterAccess;
    USERESSCHEDULER = OsUseResScheduler;
    SAVEFPUREG = OsSaveFpuReg;
    INTERRUPTBASE = OsInterruptBaseAddress;
    STACKSIZE = OsStackSize;
    [ DEFAULTFPSRVALUE = OsDefaultFPSRValue; ]
    STARTUPHOOK = OsStartupHook;
    SHUTDOWNHOOK = OsShutdownHook;
    POSTTASKHOOK = OsPostTaskHook;
    PRETASKHOOK = OsPreTaskHook;
    ERRORHOOK = OsErrorHook;
    PROTECTIONHOOK = OsProtectionHook;
    [ MEMORYAREA = OsMemoryAreaNameRef {
        ATTRIBUTE = OsMemoryAreaAttribute;
    }; ]
};
```

(1) Identifier "*OsOS*"
Specifies the OS identifier.
Only a name can be specified as *OsOS*.

(2) Maximum number of cores "*OsNumberOfCores*"
Specifies the maximum number of cores that can be controlled by the RV850.
Only 0x1 can be specified as *OsNumberOfCores*.

Remark The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 65535. In the RV850, however, only 0x1 can be specified for this item.

[If omitted:]
Processing is performed assuming that 0x1 is specified.

(3) Scalability class "*OsScalabilityClass*"
Specifies the scalability class of the RV850.
Only SC1, SC3, or AUTO can be specified as *OsScalabilityClass*.

SC1: Scalability class 1
SC3: Scalability class 3
AUTO: The configurator selects the most appropriate scalability class in accordance with the content of the CF file.

Remark 1. The AUTOSAR specifications prescribe that the keywords that can be specified for this item are SC1, SC2, SC3, and SC4. In the RV850, however, SC2 and SC4 are not supported.

Remark 2. The AUTOSAR specifications do not prescribe the keyword AUTO. This is our original keyword added to the RV850.

[If omitted:]
Processing is performed assuming that AUTO is specified.

(4) Stack monitoring facilities "*OsStackMonitoring*"
Specifies whether to use the stack monitoring facilities when a processing program (task or interrupt service routine) performs processing.
Only TRUE or FALSE can be specified as *OsStackMonitoring*.

TRUE: Used
FALSE: Not used

Remark 1. See "3.1.2Stack monitoring facilities" and "4.1.1Stack monitoring facilities" for details about the stack monitoring facilities.

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(5) Status type "*OsStatus*"
Specifies the status type of the RV850.
Only EXTENDED can be specified as *OsStatus*.

EXTENDED: Extended status

(6) OSErrorGetServiceId "*OsUseGetServiceId*"
Specifies whether to use utility function OSErrorGetServiceId.
Only TRUE or FALSE can be specified as *OsUseGetServiceId*.

TRUE: Used
FALSE: Not used

Remark 1. The AUTOSAR specifications prescribe that whether to use utility function OSErrorGetServiceId should be specified in this item. In the RV850, however, processing is performed assuming that TRUE is specified regardless of the actual definition in this item.

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(7) OSError_*SystemService_Parameter* "*OsUseParameterAccess*"
Specifies whether to use utility function OSError_SystemService_Parameter.
Only TRUE or FALSE can be specified as *OsUseParameterAccess*.

TRUE: Used
FALSE: Not used

Remark 1. The AUTOSAR specifications prescribe that whether to use utility function OSError_SystemService_Parameter should be specified in this item. In the RV850, however, processing is performed assuming that TRUE is specified regardless of the actual definition in this item.

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(8) Scheduler resource "*OsUseResScheduler*"
Specifies whether to use a scheduler resource (identifier: RES_SCHEDULER). Only TRUE or FALSE can be specified as *OsUseResSchedule*.

TRUE: Used
FALSE: Not used

Remark 1. See "5.1.1Ceiling values" for details about the scheduler resource.

Remark 2. The AUTOSAR specifications prescribe that whether to use a scheduler resource should be specified in this item. In the RV850, however, whether to use a scheduler resource is determined according to whether identifier RES_SCHEDULER is specified in Identifier "OsResource" regardless of the definition in this item.

Remark 3. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(9) FPSR saving/restoring "*OsSaveFpuReg*"
Specifies whether to make the RV850 save and restore the floating-point configuration/status register (FPSR) when execution switches from a processing program (such as a task, an interrupt service routine, or a hook routine) to another. Only TRUE or FALSE can be specified as *OsSaveFpuReg*.

TRUE: RV850 saves and restores FPSR.
FALSE: RV850 does not save or restore FPSR.

Remark 1. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(10) Base address "*OsInterruptBaseAddress*"
Specifies the base address of the interrupt handler address table.
Only a 0x200-byte aligned value from 0x0 to 0xFFFFFE00 can be specified as *OsInterruptBaseAddress*.

Remark 1. The RV850 uses the table reference method to select an interrupt handler address.

Remark 2. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(11) System stack size "*OsStackSize*"
Specifies the stack size used by the system.
Only a 0x4-byte aligned value from 0x4 to 0xFFFFFFC can be specified for *OsStackSize*.

Remark 1. See "C.7.1System stack" for details about the size specified in this item.

Remark 2. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(12) FPSR default value "*OsDefaultFPSRValue*"
This specifies the default value to be set in the floating-point configuration/status register (FPSR) when a processing program (such as a task, an interrupt service routine, or a hook routine) is activated. Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsDefaultFPSRValue*.

Remark 1. This item can be specified only when TRUE is defined for FPSR saving/restoring "OsSaveFpuReg".

Remark 2. The value defined for FPSR "OsAppDefaultFPSRValue" has precedence for the setting of the processing programs (tasks, interrupt service routines, or hook routines) belonging to the OS-Application.

Remark 3. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

[If omitted:]
Processing is performed assuming that reset value of the target device is specified.

(13) StartupHook "*OsStartupHook*"
Specifies whether to use common hook routine StartupHook.
Only TRUE or FALSE can be specified as *OsStartupHook*.

TRUE: Used
FALSE: Not used

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(14) ShutdownHook "*OsShutdownHook*"
Specifies whether to use common hook routine ShutdownHook. Only TRUE or FALSE can be specified as *OsShutdownHook*.

TRUE: Used
FALSE: Not used

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(15) PostTaskHook "*OsPostTaskHook*"
Specifies whether to use common hook routine PostTaskHook. Only TRUE or FALSE can be specified as *OsPostTaskHook*.

TRUE: Used
FALSE: Not used

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(16) PreTaskHook "*OsPreTaskHook*"
Specifies whether to use common hook routine PreTaskHook. Only TRUE or FALSE can be specified as *OsPreTaskHook*.

TRUE: Used
FALSE: Not used

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(17) ErrorHook "*OsErrorHook*"
Specifies whether to use common hook routine ErrorHook. Only TRUE or FALSE can be specified as *OsErrorHook*.

TRUE: Used
FALSE: Not used

Remark In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(18) ProtectionHook "*OsProtectionHook*"
Specifies whether to use common hook routine ProtectionHook. Only TRUE or FALSE can be specified as *OsProtectionHook*.

TRUE: Used
FALSE: Not used

Remark 1. When FALSE is specified for this item and if a protection violation (a stack overflow, illegal memory access, or occurrence of an exception) is detected, ShutdownOS will be issued.

Remark 2. TRUE can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".
The AUTOSAR specifications prescribe that a warning should be output when SC1 is defined for Scalability class "OsScalabilityClass" and TRUE is defined for this item. In the RV850, however, fatal error E4003 will be output.

Remark 3. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(19) Memory area identifier "*OsMemoryAreaNameRef*"
Specifies the identifier of the target memory area for access protection in all OS-Applications.
Only Memory area identifier "OsSystemMemoryArea" can be specified for *OsMemoryAreaNameRef*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".

Remark 2. This item can be specified multiple times (when G3K is defined for Core identifier "OsSystemCpuCore": up to 3 times, when G3M or G3KH or G3MH is defined for Core identifier "OsSystemCpuCore": up to 7 times) as shown below.

```
MEMORYAREA = OsSystemMemoryArea11 {
    ATTRIBUTE = CODE;
};
MEMORYAREA = OsSystemMemoryArea12 {
    ATTRIBUTE = CONST;
};
MEMORYAREA = OsSystemMemoryArea13 {
    ATTRIBUTE = DATA;
};
```

Remark 3. The target memory area for access protection only in a specific OS-Application should be defined for Memory area identifier "OsAppMemoryAreaNameRef".

Remark 4. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(a) Attribute "*OsMemoryAreaAttribute*"
Specifies the attribute to assign to the memory area.
Only CODE, CONST, or DATA can be specified as *OsMemoryAreaAttribute*.

CODE: Readable and executable
CONST: Readable
DATA: Readable and writeable

Remark 1. If a processing program (a task, an interrupt service routine, or a hook routine) performs memory access other than that specified in this item, processing is performed (ProtectionHook is called or ShutdownOS is issued) in accordance with the definition in ProtectionHook "OsProtectionHook".

Remark 2. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

B.4.8 Resource information

The following items are defined as required information for implementing the RESOURCE MANAGEMENT provided by the RV850.

- Identifier "OsResource"
- Ceiling value "OsResourcePriority"
- Type "OsResourceProperty"
- OS-Application identifier "OsResourceAccessingApplication"
- Resource identifier "OsResourceLinkedResourceRef"

0 to 1023 sets of (1022 if the scheduler resource is used) resource information can be specified.

Remark In the RV850, if the counter defined in this information is not used in Task information or Interrupt service routine information, fatal error E4006 will be output.

The format for defining resource information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.11 Resource Information Definition Format

```
RESOURCE OsResource {
    [ PRIORITY = OsResourcePriority; ]
    RESOURCEPROPERTY = OsResourceProperty;
    ACCESSING_APPLICATION = OsResourceAccessingApplication;
    [ LINKEDRESOURCE = OsResourceLinkedResourceRef; ]
};
```

(1) Identifier "*OsResource*"
Specifies the resource identifier.
Only a name can be specified as *OsResource*.

Remark When using the scheduler resource, specify RES_SCHEDULER for this item. See "5.1.1Ceiling values" for details about the scheduler resource.

(2) Ceiling value "*OsResourcePriority*"
Specifies the ceiling value of the resource.
Only a value from 0 to 29, INTPRI0 to INTPRI15 (or INTPRI0 to INTPRI7 when the target device is G3K), or AUTO can be specified as *OsResourcePriority*.

Remark 1. This is an optional item. When the specification is omitted, the ceiling value is automatically calculated and assigned as described in Remark 5. Except for the case of generating the scheduler resource, it is recommended to omit the specification of the ceiling value by this item, and instead specify Resource identifier "OsTaskResourceRef" to be acquired by the task or Resource identifier "OsIsrResourceRef" to be acquired by the interrupt service routine (category 2).
See "5.1.1 Ceiling values" for details about the ceiling value.

Remark 2. The *OsResourcePriority* value should be higher than the priority of the processing programs that use this resource and equal to or lower than the highest priority of all interrupt service routines (category 2) for the OS-Application to which this resource belongs.

Remark 3. To generate the scheduler resource, RES_SCHEDULER should be defined for Identifier "OsResource", and priority 29 should be defined for *OsResourcePriority*.

Remark 4. Values 0 to 29 specified in *OsResourcePriority* are priority levels, where 0 is the lowest priority and 29 is the highest. For INTPRI*x*, INTPRI0 is the lowest priority and INTPRI15 is the highest.

Remark 5. When AUTO is specified, the highest priority in all tasks and interrupt service routines, which acquire the resource, defined in the CF file is obtained, and that priority is assigned as the ceiling value.
Consequently, if a definition is made as shown below, the OsResource1 ceiling value will be "INTPRI5".

```
TASK OsTask1 {
    ..................
    ..................
    PRIORITY = 29;
    ..................
    ..................
    RESOURCE = OsResource1;
    ..................
    ..................
};
ISR OsISR1 {
    ..................
    ..................
    PRIORITY = INTPRI5;
    ..................
    ..................
    RESOURCE = OsResource1;
    ..................
    ..................
};
RESOURCE OsResource1 {
    PRIORITY = AUTO;
};
```

Remark 6. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

[If omitted:]
Processing is performed assuming that AUTO is specified.

(3) Type "*OsResourceProperty*"
Specifies the resource type.
Only STANDARD, INTERNAL, or LINKED can be specified as *OsResourceProperty*.

STANDARD:	Normal resource
INTERNAL:	Internal resource
LINKED:	Linked resource

Remark For details of the normal resource, internal resource, and linked resource, see "5.1Overview".

(4) OS-Application identifier "*OsResourceAccessingApplication*"
Specifies the identifier of an OS-Application that defines objects (tasks, interrupt service routines, and alarms) to which access privileges to this resource should be assigned.
Only Identifier "OsApplication" can be specified as *OsResourceAccessingApplication*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".

Remark 2. When LINKED is defined for Type "OsResourceProperty", this item does not need to be defined because the same value as the OS-Application identifier for the resource defined for Resource identifier "OsResourceLinkedResourceRef" is assigned to the OS-Application identifier of this resource.

Remark 3. This item can be specified multiple times (up to 31 times) as shown below.

```
ACCESSING_APPLICATION = OsApplication1;
ACCESSING_APPLICATION = OsApplication2;
ACCESSING_APPLICATION = OsApplication3;
```

(5) Resource identifier "*OsResourceLinkedResourceRef*"
Specifies the identifier of the resource that will inherit the ceiling value.
Only Identifier "OsResource" can be specified as *OsResourceLinkedResourceRef*.

Remark This item can be specified only when LINKED is defined for Type "OsResourceProperty".

B.4.9 Schedule table information

The following items are defined as required information for implementing the SCHEDULE MANAGEMENT provided by the RV850.

- Identifier "OsScheduleTable"
- Schedule count value "OsScheduleTableDuration"
- Cyclic property "OsScheduleTableRepeating"
- OS-Application identifier "OsSchTblAccessingApplication"
- Counter identifier "OsScheduleTableCounterRef"
- Initial state "OsScheduleTableAutostart"
  - Type "OsScheduleTableAutostartType"
  - Offset count value "OsScheduleTableStartValue"
  - Application mode "OsScheduleTableAppModeRef"
- Expiry conditions/expiry action
  - Expiry count value "OsScheduleTblExpPointOffset"
  - Expiry action (task activation)
    - Task identifier "OsScheduleTableActivateTaskRef"
  - Expiry action (event mask setting)
    - Event identifier "OsScheduleTableSetEventRef"
    - Task identifier "OsScheduleTableSetEventTaskRef"

The sum of the numbers of schedule table information sets and Alarm information sets must be between 0 and 1023.

Remark 1. In the RV850, if SC3 is defined for Scalability class "OsScalabilityClass" and the counter defined in this information is not defined in the OS-Application information, fatal error E4013 will be output.

Remark 2. The AUTOSAR specifications prescribe ADJUSTABLEEXPPOINT and LOCAL_TO_GLOBAL_TIME_SYNCHRONIZATION as schedule table information items. However, the RV850 does not support them.

The format for defining schedule table information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.12 Schedule table information

```
SCHEDULETABLE OsScheduleTable {
    LENGTH = OsScheduleTableDuration;
    PERIODIC = OsScheduleTableRepeating;
    [ ACCESSING_APPLICATION = OsSchTblAccessingApplication; ]
    COUNTER = OsScheduleTableCounterRef;
    [ AUTOSTART = OsScheduleTableAutostart {
        TYPE = OsScheduleTableAutostartType;
        STARTVALU = OsScheduleTableStartValue;
        [ APPMODE = OsScheduleTableAppModeRef; ]
    }; ]
    ACTION {
        OFFSET = OsScheduleTblExpPointOffset;
        [ ACTIVATETASK {
            TASK = OsScheduleTableActivateTaskRef;
        }; ]
        [ SETEVENT {
            EVENT = OsScheduleTableSetEventRef;
            TASK = OsScheduleTableSetEventTaskRef;
        }; ]
    };
};
```

(1) Identifier "*OsScheduleTable*"
Specifies the schedule table identifier.
Only a name can be specified as *OsScheduleTable*.

(2) Schedule count value "*OsScheduleTableDuration*"
Specifies the schedule count value of the schedule table.
Only a value from the Minimum cycle value "OsCounterMinCycle" to Maximum count value "OsCounterMaxAllowedValue" plus 0x1 can be specified for *OsScheduleTableDuration*.

(3) Cyclic property "*OsScheduleTableRepeating*"
Specifies whether to assign the cyclic property to the schedule table. Only TRUE or FALSE can be specified as *OsScheduleTableRepeating*.

TRUE: Cyclic property is assigned.
FALSE: Cyclic property is not assigned.

Remark 1. When TRUE is specified in *OsScheduleTableRepeating*, the schedule table operates as a cyclic schedule table; when FALSE is specified, it operates as a one-shot schedule table.

Remark 2. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

(4) OS-Application identifier "*OsSchTblAccessingApplication*"
Specifies the identifier of an OS-Application that defines objects (tasks, interrupt service routines, and counters) to which access privileges to this schedule table should be assigned. Only Identifier "OsApplication" can be specified as *OsSchTblAccessingApplication*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".

Remark 2. The identifier of the OS-Application to which this schedule table belongs does not need to be specified as *OsSchTblAccessingApplication*.

Remark 3. This item can be specified multiple times (up to 31 times) as shown below.

```
ACCESSING_APPLICATION = OsApplication1;
ACCESSING_APPLICATION = OsApplication2;
ACCESSING_APPLICATION = OsApplication3;
```

[If omitted:]
Processing is performed assuming that access privileges to this schedule table are assigned only to the objects (tasks, interrupt service routines, and counters) defined in the OS-Application to which this schedule table belongs.

(5) Counter identifier "*OsScheduleTableCounterRef*"
Specifies the identifier of the counter to associate (the counter that holds the count for confirming whether the start conditions/expiry conditions have been met).
Only Identifier "OsCounter" can be specified as *OsScheduleTableCounterRef*.

Remark When SC3 is defined for Scalability class "OsScalabilityClass" and "a counter that does not belong to the OS-Application to which this schedule table belongs (counter belonging to another OS-Application)" is specified for this counter identifier, the identifier of the OS-Application to which the counter belongs should be defined for OS-Application identifier "OsSchTblAccessingApplication".

(6) Initial state "*OsScheduleTableAutostart*"
Specifies the initial state of the schedule table.
Only TRUE or FALSE can be specified as *OsScheduleTableAutostart*.

TRUE: Depends on StartOS parameter "*Mode*".
FALSE: STOPPED state.

Remark 1. When this item is set to "TRUE", the initial state of the schedule table changes as follows according to the value set in StartOS parameter "*Mode*".

[Values of Mode and Application mode "OsScheduleTableAppModeRef" match]

- RUNNING state (other combination defined)

[Values of Mode and Application mode "OsScheduleTableAppModeRef" do not match]

- STOPPED state

Remark 2. The format for specifying FALSE is as follows:

```
AUTOSTART = FALSE;
```

Remark 3. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

(a) Type "*OsScheduleTableAutostartType*"
Specifies the type of the schedule table.
Only ABSOLUTE or RELATIVE can be specified as *OsScheduleTableAutostartType*.

ABSOLUTE: Absolute schedule table
RELATIVE: Relative schedule table

Remark The AUTOSAR specifications prescribe that a warning should be output when an illegal value is specified for this item. In the RV850, however, fatal error E4003 will be output.

(b) Offset count value "*OsScheduleTableStartValue*"
Specifies the offset count value from the issuing of StartOS until the start of schedule counting.
The values that can be specified as *OsScheduleTableRelOffset* depend on the value specified in Type "OsScheduleTableAutostartType".

- ABSOLUTE
  Only a value from 0x0 to the value obtained by "Maximum count value "OsCounterMaxAllowedValue"- *InitialOffset*" can be specified.
- RELATIVE
  Only a value from 0x1 to the value obtained by "Maximum count value "OsCounterMaxAllowedValue"- *InitialOffset*" can be specified.

Remark *InitialOffset* indicates the minimum of Expiry count value "OsScheduleTblExpPointOffset" defined for this schedule table.

(c) Application mode "*OsScheduleTableAppModeRef*"
Specifies the application mode of the schedule table. Only Application mode "OsAppMode" can be specified for *OsScheduleTableAppModeRef*.

Remark 1. Processing is performed assuming that the default application mode OSDEFAULTAPPMODE is defined regardless of whether it is actually defined.

Remark 2. This item can be specified multiple times (up to 127 times) as shown below.

```
APPMODE = OSDEFAULTAPPMODE;
APPMODE = OsAppMode1;
APPMODE = OsAppMode2;
```

[If omitted:]
The AUTOSAR specifications prescribe that this item must not be omitted. In the RV850, however, when this item is omitted, processing is performed assuming that OSDEFAULTAPPMODE is specified.

(7) Expiry conditions/expiry action
Specifies the expiry conditions and expiry action for the schedule table.

Remark This item can be specified multiple times (up to 1023 times) as shown below.

```
ACTION {
    OFFSET = 0x7;
    ACTIVATETASK {
        TASK = OsTask2;
    };
};
ACTION {
    OFFSET = 0x3;
    SETEVENT {
        EVENT = OsEvent1;
        TASK = OsTask1;
```

```
    };
};
ACTION {
    OFFSET = 0xF;
    ACTIVATETASK {
        TASK = OsTask4;
    };
    SETEVENT {
        EVENT = OsEvent3;
        TASK = OsTask3;
    };
};
```

(a) Expiry count value "*OsScheduleTblExpPointOffset*"
Specifies the expiry count value of the schedule table.
The value that can be specified for *OsScheduleTblExpPointOffset* depends on the definition of Cyclic property "OsScheduleTableRepeating".

- TRUE
  Only 0x0 or a value from Minimum cycle value "OsCounterMinCycle" to the value obtained by "Schedule count value "OsScheduleTableDuration" - Minimum cycle value "OsCounterMinCycle"" can be specified.
- FALSE
  Only 0x0 or a value from Minimum cycle value "OsCounterMinCycle" to Schedule count value "OsScheduleTableDuration" can be specified.

Remark 1. *OsScheduleTblExpPointOffset* is the relative count value from Offset count value "OsScheduleTableStartValue" until the expiry action is executed.

Remark 2. When multiple Expiry conditions/expiry action are defined, the interval between expiry points (the timing for execution of expiry action) must be at least Minimum cycle value "OsCounterMinCycle".

(b) Expiry action (task activation)
Specifies whether to execute task activation processing (processing equivalent to ActivateTask) as the expiry action of the schedule table.

Remark This item can be specified multiple times (up to 1023 times) as shown below.
The total maximum number of processes that can be specified for each expiry condition (sum of task activation ACTIVATETASK and event mask setting SETEVENT) is 1023.

```
ACTIVATETASK {
    TASK = OsTask1;
};
ACTIVATETASK {
    TASK = OsTask2;
};
ACTIVATETASK {
    TASK = OsTask3;
};
```

<1> Task identifier "*OsScheduleTableActivateTaskRef*"
Specifies the identifier of the task to activate when the expiry conditions are met.
Only Identifier "OsTask" can be specified as *OsScheduleTableActivateTaskRef*.

Remark When SC3 is defined for Scalability class "OsScalabilityClass" and "a task that does not belong to the OS-Application to which this schedule table belongs (task belonging to another OS-Application)" is specified for this task identifier, the identifier of the OS-Application to which this schedule table belongs should be defined for OS-Application identifier "OsTaskAccessingApplication".

(c) Expiry action (event mask setting)
Specifies whether to set the event mask (processing equivalent to SetEvent) as the expiry action of the schedule table.

Remark This item can be specified multiple times (up to 1023 times) as shown below.

The total maximum number of processes that can be specified for each expiry condition (sum of task activation ACTIVATETASK and event mask setting SETEVENT) is 1023.

```
SETEVENT {
    EVENT = OsEvent1;
    TASK = OsTask1;
};
SETEVENT {
    EVENT = OsEvent2;
    TASK = OsTask2;
};
SETEVENT {
    EVENT = OsEvent2;
    TASK = OsTask2;
};
SETEVENT {
    EVENT = OsEvent3;
    TASK = OsTask3;
};
```

<1> Event identifier "*OsScheduleTableSetEventRef*"
Specifies the identifier of the event that holds the event mask to set when the expiry conditions are met.
Only Identifier "OsEvent" can be specified as *OsScheduleTableSetEventRef*.

Remark The event specified here must be assigned to the task specified by Task identifier "OsScheduleTableSetEventTaskRef".

<2> Task identifier "*OsScheduleTableSetEventTaskRef*"
Specifies the identifier of the task for which the event mask is to be set when the expiry conditions are met.
Only Identifier "OsTask" can be specified as *OsScheduleTableSetEventTaskRef*.

Remark When SC3 is defined for Scalability class "OsScalabilityClass" and "a task that does not belong to the OS-Application to which this schedule table belongs (task belonging to another OS-Application)" is specified for this task identifier, the identifier of the OS-Application to which this schedule table belongs should be defined for OS-Application identifier "OsTaskAccessingApplication".

B.4.10 Task information

The following items are defined as required information for implementing the TASK MANAGEMENT provided by the RV850.

- Identifier "OsTask"
- Maximum activation request count "OsTaskActivation"
- Initial priority "OsTaskPriority"
- Scheduling attribute "OsTaskSchedule"
- Task stack size "OsTaskStackSize"
- OS-Application identifier "OsTaskAccessingApplication"
- Event identifier "OsTaskEventRef"
- Resource identifier "OsTaskResourceRef"
- Initial state "OsTaskAutostart"
    - Application mode "OsTaskAppModeRef"
- Timing protection "OsTaskTimingProtection"

0 to 1023 sets of task information can be defined.

Remark 1. The AUTOSAR specifications do not prescribe the operation when neither the this information nor Interrupt service routine information is defined. In the RV850, however, fatal error E4011 will be output in this case.

Remark 2. In the RV850, if SC3 is defined for Scalability class "OsScalabilityClass" and the task defined in this information is not defined in the OS-Application information, fatal error E4013 will be output.

The format for defining task information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.13 Task Information Definition Format

```
TASK OsTask {
    ACTIVATION = OsTaskActivation;
    PRIORITY = OsTaskPriority;
    SCHEDULE = OsTaskSchedule;
    [ STACKSIZE = OsTaskStackSize; ]
    [ ACCESSING_APPLICATION = OsTaskAccessingApplication; ]
    [ EVENT = OsTaskEventRef; ]
    [ RESOURCE = OsTaskResourceRef; ]
    [ AUTOSTART = OsTaskAutostart {
        [ APPMODE = OsTaskAppModeRef; ]
    }; ]
    [ TIMING_PROTECTION = OsTaskTimingProtection; ]
};
```

(1) Identifier "*OsTask*"
Specifies the task identifier.
Only a name can be specified as *OsTask*.

Remark When a task is defined as follows, the value set in *OsTask* should be "OsTask1".

```
TASK ( OsTask1 ) {
    ...................
    ...................
}
```

(2) Maximum activation request count "*OsTaskActivation*"
Specifies the maximum number of activation requests that can be counted.
Only a value from 0x1 to 0x7F can be specified as *OsTaskActivation*.

Remark 1. When the Event identifier "OsTaskEventRef" is defined, only 0x1 can be specified as *OsTaskActivation*.

Remark 2. The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 0x7F.

(3) Initial priority "*OsTaskPriority*"
Specifies the initial priority of the task.
Only a value from 0 to 29 can be specified as *OsTaskPriority*.

Remark 1. Values 0 to 29 specified in *OsTaskPriority* are priority levels, where 0 is the lowest priority and 29 is the highest.

Remark 2. The AUTOSAR specifications prescribe that the maximum specifiable value for this item is 0xFFFFFFFF. In the RV850, however, the maximum specifiable value for this item is 29.

(4) Scheduling attribute "*OsTaskSchedule*"
Specifies the task scheduling attribute.
Only NON or FULL can be specified as *OsTaskSchedule*.

NON: Non-preemptive
FULL: Preemptive

Remark See "12.1Overview" for details about the scheduling attribute.

(5) Task stack size "*OsTaskStackSize*"
Specifies the stack size (in bytes) of the task.
Only a 0x4-byte aligned value from 0x4 to 0xFFFFFFC can be specified for *OsTaskStackSize*.

Remark 1. This item must always be specified when Event identifier "OsTaskEventRef" is defined.

Remark 2. See "C.7.3Task stack (extended task)" for details about the size specified in this item.

Remark 3. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

(6) OS-Application identifier "*OsTaskAccessingApplication*"
Specifies the identifier of an OS-Application that defines objects (tasks, interrupt service routines, alarms, and schedule tables) to which access privileges to this task should be assigned. Only Identifier "OsApplication" can be specified as *OsTaskAccessingApplication*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass".
The AUTOSAR specifications prescribe that a warning should be output when a value other than SC3 and SC4 is defined for Scalability class "OsScalabilityClass". In the RV850, however, fatal error E4003 will be output.

Remark 2. The identifier of the OS-Application to which this task belongs does not need to be specified as *OsTaskAccessingApplication*.

Remark 3. This item can be specified multiple times (up to 31 times) as shown below.

```
ACCESSING_APPLICATION = OsApplication1;
ACCESSING_APPLICATION = OsApplication2;
ACCESSING_APPLICATION = OsApplication3;
```

[If omitted:]
Processing is performed assuming that access privileges to this task are assigned only to the objects (tasks, interrupt service routines, and counters) defined in the OS-Application to which this task belongs.

(7) Event identifier "*OsTaskEventRef*"
Specifies the identifier of the event assigned to the task.
Only Identifier "OsEvent" can be specified as *OsTaskEventRef*.

Remark This item can be specified multiple times (up to 32 times) as shown below.

```
EVENT = OsEvent1;
EVENT = OsEvent2;
EVENT = OsEvent3;
```

[If omitted:]
Processing is performed assuming that no event is assigned to the task.

(8) Resource identifier "*OsTaskResourceRef*"
Specifies the identifier of the resource that is manipulated (acquired or released) by the task. Only Identifier "OsResource" can be specified as *OsTaskResourceRef*.

Remark 1. When SC3 is defined for Scalability class "OsScalabilityClass", the identifier of the OS-Application to which this task belongs should be defined for OS-Application identifier "OsResourceAccessingApplication".

Remark 2. This item can be specified multiple times (up to 1023 times) as shown below.
However, for the resources defined as INTERNAL in Type "OsResourceProperty", this item can be defined only one time.

```
RESOURCE = OsResource1;
RESOURCE = OsResource2;
RESOURCE = OsResource3;
```

[If omitted:]
Processing is performed assuming that the task does not manipulate (acquire or release) resources.

(9) Initial state "*OsTaskAutostart*"
Specifies the initial state of the task.
Only TRUE or FALSE can be specified as *OsTaskAutostart*.

TRUE: Depends on StartOS parameter "*Mode*".
FALSE: SUSPENDED state

Remark 1. When this item is set to "TRUE", the initial state of the task changes as follows according to the value set in StartOS parameter "*Mode*".

[Values of "*Mode*" and Application mode "OsTaskAppModeRef" match]

- READY state

[Values of "*Mode*" and Application mode "OsTaskAppModeRef" do not match]

- SUSPENDED state

Remark 2. The format for specifying FALSE is as follows:

```
AUTOSTART = FALSE;
```

Remark 3. In the ARXML format, specify 1 for this item to select the processing equivalent to TRUE or specify 0 to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

(a) Application mode "*OsTaskAppModeRef*"
Specifies the application mode of the task.
Only Application mode "OsAppMode" can be specified as *OsTaskAppModeRef*.

Remark 1. Processing is performed assuming that the default application mode OSDEFAULTAPPMODE is defined regardless of whether it is actually defined.

Remark 2. This item can be specified multiple times (up to 127 times) as shown below.

```
APPMODE = OSDEFAULTAPPMODE;
APPMODE = OsAppMode1;
APPMODE = OsAppMode2;
```

[If omitted:]
The AUTOSAR specifications prescribe that this item must not be omitted. In the RV850, however, when this item is omitted, processing is performed assuming that OSDEFAULTAPPMODE is specified.

(10) Timing protection "*OsTaskTimingProtection*"
Specifies whether to use the timing protection function when task processing is performed. Only FALSE can be specified as *OsTaskTimingProtection*.

FALSE: Not used

Remark 1. The AUTOSAR specifications prescribe that use of the timing protection function should be disabled when SC1 or SC3 is defined for Scalability class "OsScalabilityClass".
In the RV850, when an illegal value is specified for this item, fatal error E4003 will be output.

Remark 2. In the ARXML format, specify 0 for this item to select the processing equivalent to FALSE.

[If omitted:]
Processing is performed assuming that FALSE is specified.

B.4.11 System information

The following items are defined as basic information required for RV850 operation as well as information required to implement the Memory protection provided by the RV850.

- Maximum exception code "OsSystemMaxExceptionCode"
- Operating frequency "OsSystemSystemClock"
- Core identifier "OsSystemCpuCore"
- EICn for INTC1 "OsSystemINTC1EIControlAddress"
- IMRm for INTC1 "OsSystemINTC1EIMaskAddress"
- EICn for INTC2 "OsSystemINTC2EIControlAddress"
- IMRm for INTC2 "OsSystemINTC2EIMaskAddress"
- Memory area identifier "OsSystemMemoryArea"
    - Exception code "OsCounterExceptionCode"
    - Size "AreaSizeValue"
    - End address "AreaEndAddressValue"

Only one set of system information can be defined.
The format for defining system information is shown below. Strings surrounded by square brackets "[ ]" are optional items that can be omitted.

Figure B.14 System Information Definition Format

```
SYSTEM {
    MAXEXCEPTIONCODE = OsSystemMaxExceptionCode;
    SYSTEM_CLOCK = OsSystemSystemClock;
    CPUCORE = OsCpuCore;
    INTC1EICTRL = OsSystemINTC1EIControlAddress;
    INTC1EIMASK = OsSystemINTC1EIMaskAddress;
    INTC2EICTRL = OsSystemINTC2EIControlAddress;
    INTC2EIMASK = OsSystemINTC2EIMaskAddress;
    [ MEMORYAREA OsSystemMemoryArea {
        STARTADDRESS = AreaStartAddressValue;
        [ SIZE = AreaSizeValue; ]
        [ ENDADDRESS = AreaEndAddressValue; ]
    }; ]
};
```

(1) Maximum exception code "*OsSystemMaxExceptionCode*"
Specifies the maximum value of the exception codes for the EI level interrupts to be managed by the RV850.
Only a value from 0x1000 to 0x11FF can be specified as *OsSystemMaxExceptionCode*.

Remark The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(2) Operating frequency "*OsSystemSystemClock*"
Specifies the operating frequency (CPU clock in kHz) of the target device.
Only a value from 0x1 to 0xFFFFFFFF can be specified as *OsSystemSystemClock*.

Remark The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(3) Core identifier "*OsSystemCpuCore*"
Specifies the identifier of the target core for control by the RV850.
Only G3K or G3M or G3KH or G3MH can be specified as *OsSystemCpuCore*.

Remark The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(4) EIC*n* for INTC1 "*OsSystemINTC1EIControlAddress*"
Specifies the start address of the EI level interrupt control register (EIC*n*) for INTC1.
Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsSystemINTC1EIControlAddress*.

Remark 1. See the user's manual of the target device for details about the value specified in *OsSystemINTC1EIControlAddress*.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(5) IMR*m* for INTC1 "*OsSystemINTC1EIMaskAddress*"
Specifies the address of the EI level interrupt mask register (IMR*m*) for INTC1.
Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsSystemINTC1EIMaskAddress*.

Remark 1. See the user's manual of the target device for details about the value specified in *OsSystemINTC1EIMaskAddress*.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(6) EIC*n* for INTC2 "*OsSystemINTC2EIControlAddress*"
Specifies the start address of the EI level interrupt control register (EIC*n*) for INTC2.
Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsSystemINTC2EIControlAddress*.

Remark 1. See the user's manual of the target device for details about the value specified in *OsSystemINTC2EIControlAddress*.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(7) IMR*m* for INTC2 "*OsSystemINTC2EIMaskAddress*"
Specifies the address of the EI level interrupt mask register (IMR*m*) for INTC2.
Only a value from 0x0 to 0xFFFFFFFF can be specified as *OsSystemINTC2EIMaskAddress*.

Remark 1. See the user's manual of the target device for details about the value specified in *OsSystemINTC2EIMaskAddress*.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(8) Memory area identifier "*OsSystemMemoryArea*"
Specifies the identifier of the target memory area for access protection by the RV850.
Only a name can be specified as *OsSystemMemoryArea*.

Remark 1. This item can be specified only when SC3 is defined for Scalability class "OsScalabilityClass". The AUTOSAR specifications prescribe that a warning should be output when a value other than SC3 and SC4 is defined for Scalability class "OsScalabilityClass". In the RV850, however, fatal error E4003 will be output.

Remark 2. This item can be specified multiple times (up to 131 times) as shown below.

```
MEMORYAREA OsSystemMemoryArea1 {
    STARTADDRESS = 0x15000000;
    SIZE = 0x50000000;
};
MEMORYAREA OsSystemMemoryArea2 {
    STARTADDRESS = 0x30000000;
    SIZE = 0x10000000;
};
MEMORYAREA OsSystemMemoryArea3 {
    STARTADDRESS = 0x60000000;
    ENDADDRESS = 0x70000000;
};
```

(a) Start address "*AreaStartAddressValue*"
Specifies the start address of the target memory area for access protection.
Only a 0x4-byte aligned value from 0x0 to 0xFFFFFFFC or a symbol name can be specified for *AreaStartAddressValue*.

Remark The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(b) Size "*AreaSizeValue*"
Specifies the size (in bytes) of the target memory area for access protection.
Only a 0x4-byte aligned value from 0x4 to 0xFFFFFFFC can be specified for *AreaSizeValue*.

Remark 1. When End address "AreaEndAddressValue" is defined, this item cannot be specified.

Remark 2. The AUTOSAR specifications do not prescribe this item. This is our original item added to the RV850.

(c) End address "*AreaEndAddressValue*"
Specifies the end address of the target memory area for access protection.
Only a 0x4-byte aligned value from 0x4 to 0xFFFFFFFC, 0xFFFFFFFF, or a symbol name can be specified for *AreaEndAddressValue*.

Remark 1. When Size "AreaSizeValue" is defined, this item cannot be specified.

Remark 2. The AUTOSAR specifications do not prescribe this item.
This is our original item added to the RV850.

C. MEMORY FOOTPRINT

This appendix describes memory footprints.

C.1 Overview

The memory area used or managed by the RV850 is broadly classified into seven types of depending on the usage.

Table C.1 Memory Areas

Name	Description
.kernel_system	Standard code area
.kernel_interface	Interface area
.kernel_const	Constant data area (ROM)
.kernel_identifier	Constant data area
.kernel_work	Variable data area (RAM)
.kernel_stack	Stack area
.kernel_*address*	Interrupt handler address table

C.2 Standard Code Area (.kernel_system)

The memory footprint of the standard code area (.kernel_system) differs as shown below according to the type of the kernel library linked when generating the load module.

Kernel Library	Size (1 Kbytes = 1024 bytes)
libecc2extsc1.a	12.2 Kbytes
libecc2extsc1_fpu.a	12.4 Kbytes
libecc2extsc3.a	22.1 Kbytes
libecc2extsc3_fpu.a	22.3 Kbytes
libecc2extsc3_g3k.a	21.6 Kbytes

C.3 Interface Area (.kernel_interface)

The memory footprint of the interface area (.kernel_interface) differs as shown below according to the type of the kernel library linked when generating the load module.

Kernel Library	Size (1 Kbytes = 1024 bytes)
libecc2extsc1.a	0.85 Kbytes
libecc2extsc1_fpu.a	0.85 Kbytes
libecc2extsc3.a	0.92 Kbytes
libecc2extsc3_fpu.a	0.92 Kbytes
libecc2extsc3_g3k.a	0.92 Kbytes

C.4 Constant Data Area (.kernel_const)

The memory footprint of the constant data area (.kernel_const) differs as shown below according to the scalability class.

(1) Constant data area for SC1 (.kernel_const)

The following shows the formula for estimating the size (in bytes) of the constant data area (.kernel_const) when the scalability class is SC1.
In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
KERNEL_CONST =
    align4 (
        673
        + 40 * Alarm_Num
        + 12 * AlarmAutostart_Num
        + 12 * AppMode_Num
        + 16 * Counter_Num
        + 16 * CounterType_Num
        + 16 * Isr_Num
        + 4 * Resource_Num
        + 32 * ScheduleTable_Num
        + 12 * ScheduleTableAutostart_Num
        + 16 * ScheduleTblExpPointOffset_Num
        + 8 * ScheduleTableAction_Num
        + 20 * Task_Num
        + 2 * TaskAutostart_Num
        + 4 * ( SystemMaxExceptionCode - 4095 )
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
AlarmAutostart_Num	Total number of alarms started automatically in each application mode
AppMode_Num	Total number of Application mode information sets (except definition of OSDEFAULTAPPMODE)
Counter_Num	Total number of Counter information sets When no Counter information is defined, specify 1.
CounterType_Num	Total number of definitions with Type "OsCounterType" set to HARDWARE
Isr_Num	Total number of Interrupt service routine information sets When no Interrupt service routine information is defined, specify 1.
Resource_Num	Total number of Resource information sets When no Resource information is defined, specify 1.
ScheduleTable_Num	Total number of Schedule table information sets When no Alarm information or Schedule table information is defined, specify 1.
ScheduleTableAutostart_Num	Total number of schedule tables started automatically in each application mode
ScheduleTblExpPointOffset_Num	Total number of expiry count value definitions in the Schedule table information
ScheduleTableAction_Num	Total number of expiry action definitions in the Schedule table information

Keyword	Description
Task_Num	Total number of Task information sets When no Task information is defined, specify 1.
TaskAutostart_Num	Total number of tasks started automatically in each application mode
SystemMaxExceptionCode	Total number of definitions of Maximum exception code "OsSystemMaxExceptionCode"

(2) Constant data area for SC3 (.kernel_const)

The following shows the formula for estimating the size (in bytes) of the constant data area (.kernel_const) when the scalability class is SC3.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
KERNEL_CONST =
    align4 (
        1253
        + 44 * Alarm_Num
        + 12 * AlarmAutostart_Num
        + 12 * AppMode_Num
        + 148 * OsApplication_Num
        + 8 * TrustedFunctionName_Num
        + 24 * Counter_Num
        + 20 * CounterType_Num
        + 20 * Isr_Num
        + 8 * Resource_Num
        + 36 * ScheduleTable_Num
        + 12 * ScheduleTableAutostart_Num
        + 16 * ScheduleTblExpPointOffset_Num
        + 8 * ScheduleTableAction_Num
        + 24 * Task_Num
        + 2 * TaskAutostart_Num
        + 4 * ( SystemMaxExceptionCode - 4095 )
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
AlarmAutostart_Num	Total number of alarms started automatically in each application mode
AppMode_Num	Total number of Application mode information sets
OsApplication_Num	Total number of OS-Application information sets
TrustedFunctionName_Num	Total number of trusted functions defined in the OS-Application information When no trusted function is defined, specify 1.
Counter_Num	Total number of Counter information sets When no Counter information is defined, specify 1.
CounterType_Num	Total number of definitions with Type "OsCounterType" set to HARDWARE
Isr_Num	Total number of Interrupt service routine information sets When no Interrupt service routine information is defined, specify 1.
Resource_Num	Total number of Resource information sets When no Resource information is defined, specify 1.
ScheduleTable_Num	Total number of Schedule table information sets When no Schedule table information is defined, specify 1.
ScheduleTableAutostart_Num	Total number of schedule tables started automatically in each application mode
ScheduleTblExpPointOffset_Num	Total number of expiry count value definitions in the Schedule table information
ScheduleTableAction_Num	Total number of expiry action definitions in the Schedule table information

Keyword	Description
Task_Num	Total number of Task information sets When no Task information is defined, specify 1.
TaskAutostart_Num	Total number of tasks started automatically in each application mode
SystemMaxExceptionCode	Total number of definitions of Maximum exception code "OsSystemMaxExceptionCode"

C.5 Constant Data Area (.kernel_identifier)

The memory footprint of the constant data area (.kernel_identifier) differs as shown below according to the scalability class.

(1) Constant data area for SC1 (.kernel_identifier)

The following shows the formula for estimating the size (in bytes) of the constant data area (.kernel_identifier) when the scalability class is SC1.

```
KERNEL_IDENTIFIER =
      2 * Alarm_Num
      + 2 * Counter_Num
      + 4 * Event_num
      + 2 * Isr_Num
      + 2 * Resource_Num
      + 2 * ScheduleTable_Num
      + 2 * Task_Num
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
Counter_Num	Total number of Counter information sets
Event_Num	Total number of Event information sets
Isr_Num	Total number of Interrupt service routine information sets
Resource_Num	Total number of Resource information sets
ScheduleTable_Num	Total number of Schedule table information sets
Task_Num	Total number of Task information sets

(2) Constant data area for SC3 (.kernel_identifier)
The following shows the formula for estimating the size (in bytes) of the constant data area (.kernel_identifier) when the scalability class is SC3.

```
KERNEL_IDENTIFIER =
      2 * Alarm_Num
      + 2 * OsApplication_Num
      + 2 * TrustedFunctionName_Num
      + 2 * Counter_Num
      + 4 * Event_num
      + 2 * Isr_Num
      + 2 * Resource_Num
      + 2 * ScheduleTable_Num
      + 2 * Task_Num
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
OsApplication_Num	Total number of OS-Application information sets
TrustedFunctionName_Num	Total number of trusted functions defined in the OS-Application information
Counter_Num	Total number of Counter information sets
Event_Num	Total number of Event information sets
Isr_Num	Total number of Interrupt service routine information sets
Resource_Num	Total number of Resource information sets
ScheduleTable_Num	Total number of Schedule table information sets
Task_Num	Total number of Task information sets

C.6 Variable Data Area (.kernel_work)

The memory footprint of the variable data area (.kernel_work) differs as shown below according to the scalability class.

(1) Variable data area for SC1 (.kernel_work)

The following shows the formula for estimating the size (in bytes) of the variable data area (.kernel_work) when the scalability class is SC1.
In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
KERNEL_WORK =
    align4 (
        128
        + 32 * Alarm_Num
        + 16 * Counter_Num
        + 12 * CounterType_Num
        + 12 * IsrCategory_Num
        + 8 * Resource_Num
        + 32 * ScheduleTable_Num
        + 24 * Task_Num
        + PriorityBuf_Ttl
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
Counter_Num	Total number of Counter information sets
CounterType_Num	Total number of definitions with Type "OsCounterType" set to HARDWARE
IsrCategory_Num	Total number of definitions with Category "OsIsrCategory" set to 2
Resource_Num	Total number of Resource information sets
ScheduleTable_Num	Total number of Schedule table information sets
Task_Num	Total number of Task information sets
PriorityBuf_Ttl	Total of the priority buffer sizes calculated in "Priority buffers"

(2) Variable data area for SC3 (.kernel_work)

The following shows the formula for estimating the size (in bytes) of the variable data area (.kernel_work) when the scalability class is SC3.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
KERNEL_WORK =
    align4 (
        148
        + 32 * Alarm_Num
        + 8 * OsApplication_Num
        + 16 * Counter_Num
        + 32 * CounterType_Num
        + 32 * IsrCategory_Num
        + 8 * Resource_Num
        + 32 * ScheduleTable_Num
        + 32 * Task_Num
        + PriorityBuf_Ttl
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
Alarm_Num	Total number of Alarm information sets
OsApplication_Num	Total number of OS-Application information sets
Counter_Num	Total number of Counter information sets
CounterType_Num	Total number of definitions with Type "OsCounterType" set to HARDWARE
IsrCategory_Num	Total number of definitions with Category "OsIsrCategory" set to 2
Resource_Num	Total number of Resource information sets
ScheduleTable_Num	Total number of Schedule table information sets
Task_Num	Total number of Task information sets
PriorityBuf_Ttl	Total of the priority buffer sizes calculated in "Priority buffers"

C.6.1 Priority buffers

The following shows the formula for estimating the buffer size (in bytes) per priority level for the priorities defined in the Task information and the ceiling values defined in the Resource information.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
PriorityBuf =
    align4 (
        8
        + 2 * ( TaskActivation_Pri + Resource - 1)
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskActivation_Pri	Total of the values defined for Maximum activation request count "OsTaskActivation" items
Resource	Set to 1 when the priority matches the ceiling value defined in any Resource information; otherwise, set to 0.

Remark If the result of "*TaskActivation_Pri* + *Resource* - 1" is 0, the PriorityBuf (buffer size per priority level) will be 0.

C.7 Stack Area (.kernel_stack)

The memory footprint of the stack area (.kernel_stack) differs as shown below according to the scalability class.

(1) Stack area for SC1 (.kernel_stack)
The following shows the formula for estimating the size (in bytes) of the stack area (.kernel_stack) when the scalability class is SC1.

```
KERNEL_STACK =
      SystemStack
    + TaskStack_Ttl
```

The meaning of each keyword used above is as follows:

Keyword	Description
SystemStack	Value calculated in "System stack for SC1"
ExtTaskStack_Ttl	Total of the values calculated in "Task stack (extended task) for SC1"

(2) Stack area for SC3 (.kernel_stack)
The following shows the formula for estimating the size (in bytes) of the stack area (.kernel_stack) when the scalability class is SC3.

```
KERNEL_STACK =
      SystemStack
    + OsApplicationStack_Ttl
    + ExtTaskStack_Ttl
```

The meaning of each keyword used above is as follows:

Keyword	Description
SystemStack	Value calculated in "System stack for SC3"
OsApplicationStack_Ttl	Total of the values calculated in "OS-Application stack"
ExtTaskStack_Ttl	Total of the values calculated in "Task stack (extended task) for SC3"

C.7.1 System stack

The memory footprint of the system stack differs as shown below according to the scalability class.

(1) System stack for SC1

The following shows the formula for estimating the size (in bytes) of the system stack when the scalability class is SC1.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary, and "Max (*x*, *y*, *z*)" means comparing *x*, *y*, and *z* and taking the largest value.

```
SystemStack =
    align4 (
        44
        + ErrorHookStack_Siz
        + ShutdownHokkStack_Siz
        + AlarmCallbackStack_Siz
        + Max (
            TaskStack_Ttl + MAX (
                8 + PostTaskHookStackSiz,
                8 + PreTaskHookStack_Siz
            ),
            20 + StartupHookStack_Siz,
            IdleHandlerStack_Siz
        )
        + MAX (
            92 + IsrStack_Max * Nest_Count,
            SystemServiceStack_Max
        )
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
ErrorHookStack_Siz	Stack size necessary for the processing of common hook routine ErrorHook
ShutdownHookStack_Siz	Stack size necessary for the processing of common hook routine ShutdownHook
AlarmCallbackStack_Siz	Stack size necessary for the processing of the alarm callback
TaskStack_Ttl	Total of the values calculated in "Task stack (basic task) for SC1"
PostTaskHookStack_Siz	Stack size necessary for the processing of common hook routine PostTaskHook
PreTaskHookStack_Siz	Stack size necessary for the processing of common hook routine PreTaskHook
StartupHookStack_Siz	Stack size necessary for the processing of common hook routine StartupHook
IdleHandlerStack_Siz	Stack size necessary for the processing of the idle handler
IsrStackMax	Maximum of the values calculated in "Interrupt service routine stack (category 2) for SC1"
Nest_Count	Maximum nesting level of interrupt service routines (category 2)
SystemServiceStack_Max	164 (Stack size necessary for the system service issued by the extended task)

(a) Task stack (basic task) for SC1

The following shows the formula for estimating the size (in bytes) of the task stack (basic task) when the scalability class is SC1.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
TaskStack =
    align4 (
        164
        + TaskStack_Siz
        + AlarmCallbackStack_Siz
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskStack_Siz	Stack size necessary for the processing of the basic stack
AlarmCallbackStack_Siz	Stack size necessary for the processing of the alarm callback

(b) Interrupt service routine stack (category 2) for SC1

The following shows the formula for estimating the size (in bytes) of the interrupt service routine stack (category 2) when the scalability class is SC1.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
IsrStack =
    align4 (
        256
        + IsrStack_Siz
        + AlarmCallbackStack_Siz
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
IsrStack_Siz	Stack size necessary for the processing of the interrupt service routine (category 2)
AlarmCallbackStack_Siz	Stack size necessary for the processing of the alarm callback

(2) System stack for SC3

The following shows the formula for estimating the size (in bytes) of the system stack when the scalability class is SC3.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary, and "Max (*x*, *y*, *z*)" means comparing *x*, *y*, and *z* and taking the largest value.

```
SystemStack =
    align4 (
        Max (
            Max (
                320,
                264 + ProtectionHookStack_Siz + ErrorHookStack_Siz,
                104 + StartupHookStack_Siz + ErrorHookStack_Siz
            )
            + Max (
                104,
                44 + ShutdownHookStack_Siz
            ),
            IdleHandlerStack_Siz
        )
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
ProtectionHookStack_Siz	Stack size necessary for the processing of common hook routine ProtectionHook
ErrorHookStack_Siz	Stack size necessary for the processing of common hook routine ErrorHook
StartupHookStack_Siz	Stack size necessary for the processing of common hook routine StartupHook
ShutdownHookStack_Siz	Stack size necessary for the processing of common hook routine ShutdownHook
IdleHandlerStack_Siz	Stack size necessary for the processing of the idle handler

C.7.2 OS-Application stack

The following shows the formula for estimating the size (in bytes) of the OS-Application stack. In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary, and "Max (*x*, *y*, *z*)" means comparing *x*, *y*, and *z* and taking the largest value.

```
OsApplicationStack =
    align4 (
        32
        + AppShutdownHookStack_Siz
        + Max (
            TaskStack_Ttl + MAX (
                48 + PostTaskHookStack_Siz,
                48 + PreTaskHookStack_Siz,
            ),
            32 + AppStartupHookStack_Siz
        )
        + MAX (
            104 + ErrorHookStack_Siz,
            216 + AppErrorHookStack_Siz
        )
        + MAX (
            140 + IsrStack_Max * Nest_Count,
            SystemServiceStack_Max
        )
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
AppShutdownStackHook_Siz	Stack size necessary for the processing of OS-Application-specific hook routine Shutdown_*OsApplication*
TaskStack_Ttl	Total of the values calculated in "Task stack (basic task) for SC3"
PostTaskHookStack_Siz	Stack size necessary for the processing of common hook routine PostTaskHook
PreTaskHookStack_Siz	Stack size necessary for the processing of common hook routine PreTaskHook
AppStartupHookStack_Siz	Stack size necessary for the processing of OS-Application-specific hook routine StartupHook_*OsApplication*
ErrorHookStack_Siz	Stack size necessary for the processing of common hook routine ErrorHook
AppErrorHookStack_Siz	Stack size necessary for the processing of OS-Application-specific hook routine ErrorHook_*OsApplication*
IsrStackMax	Maximum of the values calculated in "Interrupt service routine stack (category 2) for SC3"
Nest_Count	Maximum nesting level of interrupt service routines (category 2)
SystemServiceStack_Max	164 (Stack size necessary for the system service issued by the extended task)

(a) Task stack (basic task) for SC3

The following shows the formula for estimating the size (in bytes) of the task stack (basic task) when the scalability class is SC3.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
TaskStack =
    align4 (
        164
        + TaskStack_Siz
        + 24 * TrustedFunctionNest_Max
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskStack_Siz	Stack size necessary for the processing of the basic task
TrustedFunctionNest_Max	Maximum nesting level of trusted functions

(b) Interrupt service routine stack (category 2) for SC3

The following shows the formula for estimating the size (in bytes) of the interrupt service routine stack (category 2) when the scalability class is SC3.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
IsrStack =
    align4 (
        304
        + IsrStack_Siz
        + 24 * TrustedFunctionNest_Max
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
IsrStack_Siz	Stack size necessary for the processing of the interrupt service routine (category 2)
TrustedFunctionNest_Max	Maximum nesting level of trusted functions When the OS-Application is non-trusted and the maximum nesting level is 0, set this value to 1.

C.7.3 Task stack (extended task)

The memory footprint of the task stack (extended task) differs as shown below according to the scalability class.

(1) Task stack (extended task) for SC1

The following shows the formula for estimating the size (in bytes) of the task stack (extended task) when the scalability class is SC1.

In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
ExtTaskStack =
    align4 (
        TaskStack_Siz
        + TaskContext_Siz
        + IsrContext_Siz
        + SystemServiceFrame_Siz
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskStack_Siz	Stack size necessary for the processing of the extended task
TaskContext_Siz	Set to 0 when NON is defined for Scheduling attribute "OsTaskSchedule"; set to 48 when FULL is defined.
IsrContext_Siz	124 (Stack size necessary for the interrupt processing occurring during execution of the extended task)
SystemServiceFrame_Siz	8 (Stack size necessary for the system service issued by the extended task)

(2) Task stack (extended task) for SC3
The memory footprint of the task stack (extended task) for SC3 differs as shown below depending on whether the task belongs to a trusted OS-Application or a non-trusted OS-Application.

(a) Trusted OS-Application
The following shows the formula for estimating the size (in bytes) of the task stack (extended task) for SC3 when the reliability of the OS-Application is defined as trusted.
In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
ExtTaskStack =
    align4 (
        TaskStack_Siz
        + TaskContext_Siz
        + IsrContext_Siz
        + SystemServiceFrame_Siz
        + 24 * TrustedFunctionNest_Max
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskStack_Siz	Stack size necessary for the processing of the extended task
TaskContext_Siz	Set to 0 when NON is defined for Scheduling attribute "OsTask-Schedule"; set to 48 when FULL is defined.
IsrContext_Siz	128 (Stack size necessary for the interrupt processing occurring during execution of the extended task)
SystemServiceFrame_Siz	12 (Stack size necessary for the system service issued by the extended task)
TrustedFunctionNest_Max	Maximum nesting level of trusted functions

(b) Non-trusted OS-Application
The following shows the formula for estimating the size (in bytes) of the task stack (extended task) for SC3 when the reliability of the OS-Application is defined as non-trusted.
In the formula, "align4 (*x*)" means the result of aligning the value "*x*" to a 4-byte boundary.

```
ExtTaskStack =
    align4 (
        140
        + TaskStack_Siz
        + 24 * TrustedFunctionNest_Max
    )
```

The meaning of each keyword used above is as follows:

Keyword	Description
TaskStack_Siz	Stack size necessary for the processing of the extended task
TrustedFunctionNest_Max	Maximum nesting level of trusted functions

C.8 Interrupt Handler Address Table (.kernel_address)

The following shows the formula for estimating the size (in bytes) of the interrupt handler address table (.kernel_*address*).

```
IntHdrAdrTbl =
    4 * (SystemMaxExceptionCode - 4095)
```

The meaning of each keyword used above is as follows:

Keyword	Description
SystemMaxExceptionCode	Total number of definitions of Maximum exception code "OsSystemMaxExceptionCode"

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Apr 25, 2014	-	First Edition issued
1.01	Sep 30, 2014	10	[1.4 Execution Environment] Changed the sentence in the "Remark 3." - "(2) Peripheral controllers" as follows: ... in scalability class SC3 conforming to ... -->... ... in scalability class SC3 (only in G3M core) conforming to ...
		10	[1.4 Execution Environment] Changed the sentence in the "SPID bit of machine configuration (MCFG0)" - "Table 1.1 OS Reerved Resource Occupied by RV850" as follows: SC3 -->SC3 (only in G3M core).
		10	[1.4 Execution Environment] Changed the sentence in the "Table 1.1 OS Reserved Resource Occupied by RV850" as follows: EITB*n* bit of EI level interrupt control register (EIC*n*) of the interrupt controller (INTC1) EITB*n* bit of EI level interrupt control register (EIC*n*) of the interrupt controller (INTC2) -->EITB*n* bit and EIP3*n*-0*n* bit of EI level interrupt control register (EIC*n*) of the interrupt controller (INTC1 or INTC2) corresponding to the interrupts which are defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"
		12	[1.5.1 Object release version] Changed the sentence in the "(9) <rx_root>\{SC1, SC3}\lib\<Target name>\r32" as follows: libecc2extsc3.a: ECC2, extended status, SC3, FPU not supported libecc2extsc3_fpu.a: ECC2, extended status, SC3, FPU supported -->libecc2extsc3.a: ECC2, extended status, SC3, FPU not supported (only in G3M core) libecc2extsc3_fpu.a: ECC2, extended status, SC3, FPU supported (only in G3M core)
		12	[1.5.1 Object release version] Added the sentence in the "(9) <rx_root>\{SC1, SC3}\lib\<Target name>\r32" below. libecc2extsc3_g3k.a: ECC2, extended status, SC3, FPU not supported (only in G3K core)
		14	[1.5.2 Source release version] Changed the title in the "(10)" as follows: <rx_root>\src\os\trace --><rx_root>\{SC1, SC3}\src\os\trace

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	14	[1.5.2 Source release version] Changed the sentence in the "(10) <rx_root>\{SC1, SC3}\src\os\trace" as follows: The source files (for scalability class SC3) of ... -->The source files of ...
		15	[2.2 Writing User-Own Coding Modules] Changed the sentence in this section as follows: ... , and provided as sample source files (excent.850). -->... , and provided as sample source files (direct_vector.850, excent.850).
		15	[2.2 Writing User-Own Coding Modules] Changed the item title in the "(1)" as follows: FE level exception entry process -->Entry process (direct branch method exception vector)
		15	[2.2 Writing User-Own Coding Modules] Changed the sentence in the "(1) Entry process (direct branch method exception vector)" as follows: The FE level exception entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (process corresponding to the type of the generated FE level exception: boot process, process corresponding to a protection exception) to the address of the handler to which the device forcibly transfers control when an FE level exception occurs. -->The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level Interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode", etc.) has been generated.
		15	[2.2 Writing User-Own Coding Modules] Changed the item title in the "(2)" as follows: Default interrupt service routine -->Exception/interrupt safety measure process
		15	[2.2 Writing User-Own Coding Modules] Changed the sentence in the "(2) Exception/interrupt safety measure process" as follows: A default interrupt service routine is a routine dedicated to the interrupt process that is called when an EI level interrupt that is not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode" has been generated. -->The exception/interrupt safety measure process is a routine dedicated to the safety measure process that is called from entry process when FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	16	[2.2.1 Generating user-own libraries] Changed the sentence in this item as follows: ... for the source files generated in ... -->... ... for the source files (exception/interrupt safety measure process: excent.850) generated in ...
		16	[2.3 Writing Processing Programs] Changed the sentence in the "(3) Interrupt service routine" as follows: ... is called when an interrupt occurs. -->... ... is called when an EI level interrupt (defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode") generates.
		17	[2.5 Writing the Linker Directive File] Changed the sentence in this section below: Write the linker directive file described the information required for fixing the memory allocations (e.g. and address information) executed by the link editor. -->... Write the linker directive file required to fix the memory allocation executed by the link editor.
		17	[2.5 Writing the Linker Directive File] Added the sentence in the "Table 2.1 Prescribed Allocation Locations" below. .kernel_*address* A ROM Interrupt handler address table
		17	[2.5 Writing the Linker Directive File] Added the "Remark 3." in this section below. The user does not need to code the definition related to .kernel_*address* for the linker directive file, because the configurator outputs to the ENTRY file about the location of the interrupt handler address table .kernel_*address*. The "*address*" of .kernel_*address* was calculated via "Base address "OsInterruptBaseAddress" + 4 * Interrupt channel number".
		18	[2.6 Generation of Load Module] Added the "Remark 5." in this section below. Registering common hook routines as library routines is prohibited in the RV850.
		18	[2.6 Generation of Load Module] Changed the sentence in the "Remark 7." as follows: ... with kernel libraries libecc2stdsc1_fpu.a, libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a which support FPU. -->... ... with kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a which support FPU.
		20	[3.1.4 Processing in tasks] Changed the sentence in the "Remark" - "(2) Saving/Restoring FPSR" as follows: ... when FPU-supporting kernel libraries libecc2stdsc1_fpu.a, libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked. -->... ... when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	23	[4.1 Overview] Changed the sentence in this section as follows: Consequently, routines dedicated to interrupt processing of interrupts other than those defined in Exception code "OsIsrExceptionCode" and Exception code "OsCOunterExceptionCode" are processes not managed by the RV850. -->Consequently, routines (boot process, exception/interrupt safety measure process, etc.) dedicated to processing of interrupts that are not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode" are processes not managed by the RV850.
		23	[4.2 Boot Process] Changed the item title in the "(4)" as follows: ... of the machine configuration register (MCFG0) --> ... of the machine configuration register (MCFG0) (only in G3M core)
		25	[4.3 Interrupt Service Routines] Changed the sentence in this section as follows: Interrupt service routines are dedicated to interrupt processes that are activated when an interrupt is generated. --> This is a routine dedicated to the interrupt process that is called when an EI level interrupt (defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode") generates.
		27	[4.3.1 Processing in interrupt service routines] Added the sentence in the "(d) Interrupt acceptance" - "(1) Category 1" below. Therefore, when a category 1 interrupt occurs, the device manipulates the ID bit in the program status word (PSW) to disable the acceptance of interrupts.
		27	[4.3.1 Processing in interrupt service routines] Added the "Remark 1." in the "(e) Issuing system services" - "(1) Category 1" below. The RV850 does not guarantee correct operation when an unallowable system service is issued from an interrupt service routine.
		27	[4.3.1 Processing in interrupt service routines] Changed the sentence in the "Remark" - "(b) Saving/Restoring FPSR" - "(2) Category 2" as follows: ... when FPU-supporting kernel libraries libecc2stdsc1_fpu.a, libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked. --> ... when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.
		27	[4.3.1 Processing in interrupt service routines] Changed the sentence in the "(c) Stack switching" - "(2) Category 2" as follows: ... , it switches to the system stack defined in Stack size "OsStackSize". --> ... , it switches to the system stack defined in Stack size "OsStackSize" or the OS-application stack defined in Stack size "OsAppStackSize".

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	28	[4.3.3 Termination of interrupt service routines] Changed the sentence in the "(2) For category 2 and scalability class SC1" as follows: Control is returned to the processing program in which an interrupt was generated. -->The scheduler is activated.
		29	[4.3.3 Termination of interrupt service routines] Changed the sentence in the "(3) For category 2 and scalability class SC3" as follows: Control is returned to the processing program in which an interrupt was generated. -->The scheduler is activated.
		29	[4.3.3 Termination of interrupt service routines] Changed the sentence in the "Remark 2." as follows: ... when FPU-supporting kernel libraries libecc2stdsc1_fpu.a, libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked. -->... when FPU-supporting kernel libraries libecc2extsc1_fpu.a, and libecc2extsc3_fpu.a are linked.
		29	[4.5.1 Entry process (direct branch method exception vector)] Changed the sentence in this section as follows: The FE level exception entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (process corresponding to the type of the generated FE level exception: boot process, process corresponding to a protection exception) to the address of the handler to which the device forcibly transfers control when an FE level exception occurs. The basic form for coding the FE level exception entry process in the assembly language is shown below. -->The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated. The basic form for coding the entry process in the assembly language is shown below.

		29	[4.5.1 Entry process (direct branch method exception vector)]
			Replaced the basic form coding in the "[SC1, RBASE/EBASE: 0x0]" as follows:
			
			`.globl _entry0000`
			`.globl _entry0010`
			`.........`
			`.globl _entry0100`
			`.........`
			
			`.org 0x00000000`
			`_entry0000:`
			`jr _boot`
			
			`.org 0x00000010`
			`_entry0010:`

Rev.	Date	Description	
		Page	Summary

1.01	Sep 30, 2014		`    jr      __kernel_e_IllegalExcEntry`
			
			`    .........`
			
			`    .org    0x00000100`
			`_entry0100:`
			`    jr      __kernel_e_IllegalExcEntry`
			
			`    .........`

		30	[4.5.1 Entry process (direct branch method exception vector)]
			Replaced the basic form coding in the "[SC3, RBASE/EBASE: 0x0]" as folloes:
			
			`    .globl  _entry0000`
			`    .globl  _entry0010`
			`    .globl  _entry0020`
			`    .........`
			`    .globl  _entry0090`
			`    .globl  _entry00A0`
			`    .globl  _entry00B0`
			`    .........`
			`    .globl  _entry0100`
			`    .........`
			
			`    .org    0x00000000`
			`_entry0000:`
			`    jr      _boot`
			
			`    .org    0x00000010`
			`_entry0010:`
			`    jr      __kernel_e_ProtectEntry`
			
			`    .org    0x00000020`
			`_entry0020:`
			`    jr      __kernel_e_IllegalExcEntry`
			
			`    .........`
			
			`    .org    0x00000090`
			`_entry0090:`
			`    jr      __kernel_e_ProtectEntry`
			
			`    .org    0x000000A0`
			`_entry00A0:`
			`    jr      __kernel_e_ProtectEntry`
			
			`    .org    0x000000B0`
			`_entry00B0:`
			`    jr      __kernel_e_IllegalExcEntry`
			
			`    .........`
			
			`    .org    0x00000100`
			`_entry0100:`
			`    jr      __kernel_e_IllegalExcEntry`
			
			`    .........`

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	30	[4.5.1 Entry process (direct branch method exception vector)] Changed the sentence in the "Remark 1." as follows: The entry process associated with EI level interrupts is output to the ENTRY file. -->The user does not need to code the entry process associated with EI level interrupts that are defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode", because the configurator outputs to the ENTRY file about the entry process.
		31	[4.5.2 Exception/interrupt measure process] Changed the sentence in this section as follows: A default interrupt service routine is a dedicated interrupt process that is called when an EI level interrupt is generated corresponding to an exception code that is not defined in Exception code "OsIsrExceptionCode or Exception code "OsCounterExceptionCode". The basic form for coding a default interrupt service routine in the C language is shown below. -->The exception/interrupt safety measure process is a routine dedicated to the safety measure process that is called from entry process when FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated. The basic form for coding an exception/interrupt safety measure process in the C language is shown below.
		31	[4.5.2 Exception/interrupt measure process] Changed the sentence in the "Remark" as follows: A default interrupt service routine is prepared in the RV850. Consequently, if _kernel_e_IllegalExcEntry is not coded, the default interrupt service routine (the process to issue ShutdownOS in which E_OS_SYS_ILLEGAL_EXCEPTION has been specified for parameter *Error*) will be called. -->An exception/interrupt safety measure process is prepared in the RV850. Consequently, event _kernel_e_IllegalExcEntry is not coded, if the branch process to the exception/interrupt safety measure process was assigned to entry process, and, in the case of the operand of SYSCALL instruction is an illegal value, the exception/interrupt safety measure process (the process to issue ShutdownOS in which E_OS_SYS_ILLEGAL_EXCEPTION has been specified for parameter *Error*) will be called.
		35	[8.1 Overview] Changed the sentence in this section as follows: The RV850 provides alarm management functions as a mechanism to operate synchronous with change of counter values. -->The RV850 provides alarm management functions as a mechanism to perform processing in synchronization with the change of counter values.
		37	[9.1 Overview] Changed the sentence in this section as follows: The RV850 provides schedule table management functions as a mechanism to operate synchronous with change of counter values. -->

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		The RV850 provides schedule table management functions as a mechanism to perform processing in synchronization with the change of counter values.
		39	[10.1.3 Memory protection] Added the sentence in this section bellow. The memory areas that are subject to monitoring are defined in Memory area identifier "OsSystemMemoryArea", and the type (OS-application specific, common for system) is defined in Memory area identifier "OsAppMemoryAreaNameRef", Memory area identifier "OsMemoryAreaNameRef".
		39	[10.1.3 Memory protection] Added the "Remark" in this section below. For the stack area, no definitions are necessary in Memory area identifier "OsSystemMemoryArea", Memory area identifier "OsAppMemoryAreaNameRef", or Memory area identifier "OsMemoryAreaNameRef".
		40	[10.2.1 Processing in trusted functions] Changed the sentence in the "(2) Saving/Restoring FPSR" as follows: Since the RV850 characterizes a trusted function as an extension of the processing program that issued CallTrustedFunction, the save/restore processes of the floating-point configuration/status register (FPSR) are not executed. Consequently, it is necessary to code the save/restore processes of FPSR in order to change the contents of FPSR explicitly. -->> The FPSR value is changed to the value defined in FPSR "OsAppDefaultFPSRValue" or FPSR default value "OsDefaultFPSRValue". Consequently, it is not necessary to code the FPSR save/restore processes when TRUE is defined in FPSR saving/restoring "OsSaveFpuReg".
		40	[10.2.1 Processing in trusted functions] Added the "Remark" in the "(2) Saving/Restoring FPSR" below. The FPSR save/restore processes are done only when kernel library libecc2extsc1_fpu.a or libecc2extsc3_fpu.a, which supports the FPU, is linked.
		41	[10.3 OS-Application-Specific Hook Routines] Added the "Remark 2." in the "(1) StartupHook_*OsApplication*" below. When StartupHook_OsApplication "OsAppStartupHook" is specified as TRUE in multiple sets of OS-Application information, StartupHook_*OsApplication* is called in the order of appearance in the CF file.
		41	[10.3 OS-Application-Specific Hook Routines] Changed the sentence in the "Remark 2." - "(2) ShutdownHook_*OsApplication*" as follows: ... is the default interrupt service routine provided by the RV850, ... -->> ... is the exception/interrupt safety measure process, ...
		41	[10.3 OS-Application-Specific Hook Routines] Added the "Remark 3." in the "(2) ShutdownHook_*OsApplication*" below. When parameter *Fatalerror* is set to E_OS_SYS_ILLEGAL_EXCEPTION, the EIIC or FEIC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcCode in this hook routine, or the EIPC or FEPC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcPC.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	41	[10.3 OS-Application-Specific Hook Routines] Added the "Remark 4." in the "(2) ShutdownHook_*OsApplication*" below. When ShutdownHook_OsApplication "OsAppShutdownHook" is specified as TRUE in multiple sets of OS-Application information, ShutdownHook_*OsApplication* is called in the order of appearance in the CF file.
		42	[10.3 OS-Application-Specific Hook Routines] Changed the sentence in the "Remark 4." - "(3) ErrorHook_*OsApplication*" as follows: ... , ErrorHook_*OsApplication* is not called again. -->... ... , ErrorHook and ErrorHook_*OsApplication* is not called again.
		42	[10.3.1 Processing in OS-Application-specific hook routines] Changed the sentence in the "Remark 1." - "(4) Interrupt acceptance" as follows: In the RV850, It is prohibited to change the category 2 interrupt acceptance status to enabled from within an OS-Application-specific hook routine. -->... It is prohibited to explicitly manipulate the category 2 interrupt acceptance status from within an OS-Application-specific hook routine.
		42	[10.3.1 Processing in OS-Application-specific hook routines] Added the "Remark 2." in the "(4) Interrupt acceptance" below. When the RV850 transfers control to ShutdownHook_*OsApplication*, the ID bit of PSW is manipulated as well as the PM*n* bits of PMR to disable the acceptance of interrupts.
		44	[11.2 Common Hook Routines] Changed the sentence in the "Remark 1." - "(2) ShutdownHook" as follows: ... is the default interrupt service routine provided by the RV850, ... -->... ... is the exception/interrupt safety measure process, ...
		44	[11.2 Common Hook Routines] Added the "Remark 2." in the "(2) ShutdownHook" below. When parameter *Fatalerror* is set to E_OS_SYS_ILLEGAL_EXCEPTION, the EIIC or FEIC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcCode in this hook routine, or the EIPC or FEPC register value can be obtained by issuing OSIllegalException_SystemRegister_ExcPC.
		45	[11.2 Common Hook Routines] Changed the sentence in the "Remark 2." - "(5) ErrorHook" as follows: ... , ErrorHook is not called again. -->... ... , ErrorHook and ErrorHook_*OsApplication* is not called again.
		45	[11.2 Common Hook Routines] Changed the sentence in the "Remark 2." - "(6) ProtectionHook" as follows: The value of the FE-level-exception accepted state saving register (FEPC) when a protection violation was detected is set in the area specified by parameter *adr*. -->... In the area specified by parameter *adr*, "0" is stored if parameter *Fatalerror* is

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		E_OS_STACKFAULT or the value of "the status save register when acknowledging FE-level-interrupt (FEPC)" when a protection violation was detected if Fatalerror is E_OS_PROTECTION_MEMORY.
		45 - 46	[11.2 Common Hook Routines] Changed the sentence in the "[PRO_TERMINATETASKISR (0x1)]" - "Remark 3." - "(6) ProtectionHook" as follows: If a task detects a protection violation, the task is shifted to SUSPENDED state, and then the scheduler is activated. If a processing program other than a task detected the protection violation, control is returned to the task that called the processing program, and processing of the task is resumed. -->The process will differ as follows depending on the type of processing program that has generated a protection violation. [Task] Shifts the task to SUSPENDED state. Releases the resources that have been acquired by the task. Issues EnableAllInterrupts if the task has issued DisableAllInterrupts Issues ResumeAllInterrupts if the task has issued SuspendAllInterrupts. Issues ResumeOSInterrupts if the task has issued SuspendOSInterrupts. Activates the scheduler. [Interrupt service routine] Releases the resources that have been acquired by the task. Issues EnableAllInterrupts if the task has issued DisableAllInterrupts Issues ResumeAllInterrupts if the task has issued SuspendAllInterrupts. Issues ResumeOSInterrupts if the task has issued SuspendOSInterrupts. Activates the scheduler. [Others] Same processing as PRO_TERMINATEAPPL.
		46	[11.2 Common Hook Routines] Changed the sentence in the "[PRO_TERMINATEAPPL (0x2)]" - "Remark 3." - "(6) ProtectionHook" as follows: TerminateApplication is issued (restart option: NO_RESTART) for the OS-Application to which the processing program that has caused the protection violation belongs. -->Executes the process equivalent to TerminateApplication (with the restart option set to NO_RESTART) for the OS-Application to which the processing program that has generated a protection violation belongs. If no OS-Application is in APPLICATION_ACCESSIBLE state or Task identifier "OsRestartTask" has not been defined, ShutdownOS (with the inherited data set to *Fatalerror*) is issued.
		46	[11.2 Common Hook Routines] Changed the sentence in the "[PRO_SHUTDOWN (0x4)]" - "Remark 3." - "(6) ProtectionHook" as follows: ShutdownOS is issued (inherited data: Fatalerror) for the OS-Application to which the processing program that has caused the protection violation belongs. -->ShutdownOS (inherited data: *Fatalerror*) is issued.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014	46	[11.2 Common Hook Routines] Changed the sentence in the "[PRO_TERMINATEAPPL_RESTART (0x12)]" - "Remark 3." - "(6) ProtectionHook" as follows: TerminateApplication is issued (restart option: RESTART) for the OS-Application to which the processing program that has caused the protection violation belongs. -->` Executes the process equivalent to TerminateApplication (with the restart option set to RESTART) for the OS-Application to which the processing program that has generated a protection violation belongs. If no OS-Application is in APPLICATION_ACCESSIBLE state or Task identifier "OsRestartTask" has not been defined, ShutdownOS (with the inherited data set to *Fatalerror*) is issued.
		46	[11.2 Common Hook Routines] Added the sentence in the "Remark 3." - "(6) ProtectionHook". [Others] ShutdownOS (inherited data: *Fatalerror*) is issued.
		46	[11.2 Common Hook Routines] Added the "Remark 4." in the "(6) ProtectionHook". The parameter *adr* is not specified by the AUTOSAR specifications. This parameter have been uniquely added to the RV850.
		46	[11.2.1 Processing in common hook routines] Added the "Remark" in the "(1) Saving/Restoring registers" below. When the RV850 transfers control to ProtectionHook, the working register for use with FE levels (FEWR) is used without being saved and restored. Consequently, after control moves to ProtectionHook, the FEWR value becomes undefined.
		46	[11.2.1 Processing in common hook routines] Changed the sentence in the "Remark 2." - "(4) Interrupt acceptance" as follows: When the RV850 transfers control to ProtectionHook, in addition to ... -->` When the RV850 transfers control to ShutdownHook and ProtectionHook, in addition to ...
		48	[12.1 Overview] Changed the sentence in the "(2) Preemptive" as follows: System service issued in which a task state transition may occur An instruction to return from a category 2 interrupt service routine is issued Schedule issued -->` ActivateTask issued TerminateTask issued ChainTask issued Schedule issued ReleaseResource issued SetEvent issued WaitEvent issued TerminateApplication issued Protection exception (system error exception, memory protection exception, privileged instruction exception) occurred An instruction to return from a category 2 interrupt service routine is issued

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		Alarm or schedule table expired
		50	[13.1 Overview] Added the "Remark" in this section below: The RV850 does not guarantee correct operation if the user manipulates the EI level interrupt mask register (IMR*m*) to enable the acceptance of the EL-level interrupts defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode" before the hook routines are called through StartOS.
		51	[13.2 Entry Process (Direct Branch Method Exception Vector)] Changed the sentence in this section as follows: The FE level exception entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (process corresponding to the type of the generated FE level exception: boot process, process corresponding to a protection exception) to the address of the handler to which the device forcibly transfers control when an FE level exception occurs. Remark See "4.5.1 FE level exception entry process" for details about the FE level exception entry process. -->The entry process is a routine dedicated to the entry process, extracted to assign the branch process to the relevant process (boot process, exception/interrupt safety measure process, etc.) when reset (RESET), FE level interrupts (FENMI, TRAP, etc.), EI level interrupts (not defined in Exception code "OsIsrExceptionCode" or Exception code "OsCounterExceptionCode"), etc. has been generated. Remark See "4.5.1 Entry process (direct branch method exception vector)" for details about the entry process.
		52	[14.1 Overview] Added the utility function in the "(10) Utility functions" below. OSIllegalException_SystemRegister_ExcCode, OSIllegalException_SystemRegister_ExcPC
		52	[14.1 Overview] Changed the sentence in the "Remark" as follows: The utility functions InitApplicationInterrupts and _kernel_fv0_InitializeIntService are not specified by the AUTOSAR specifications. -->The utility functions InitApplicationInterrupts, _kernel_fv0_InitializeIntService, OSIllegalException_SystemRegister_ExcCode and OSIllegalException_SystemRegister_ExcPC are not specified by the AUTOSAR specifications.
		53	[14.1.1 Calling of system services] Added the sentence in this section below. If a system service of Interrupt handling is issued before the system initialization process is finished, _kernel_fv0_InitializeIntService needs to be issued before the system service is issued.
		53	[14.1.1 Calling of system services] Added the "Remark 3." in this section below. If the illegal value is set when SYSCALL instruction is issued, exception/interrupt

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		safety measure process "_kernel_e_IllegalExcEntry" is called.
		54 - 55	[14.2.1 Data types] Added the macro in the "Table 14.1 Data Types" below. boolean, float32, float64, sint8, sint8_least, sint16, sint16_least, sint32, sint32_least, sint64, uint8, uint8_least, uint16, uint16_least, uint32, uint32_least, uint64
		57	[14.2.6 Exit with error (abend)] Changed the sentence in the "INVALID_ISR" - "Table 14.6 Exit with Error (Abend)" as follows: Issued from a processing program outside the scope of issue. -->Exit with error
		63	[14.2 Data Macros] Added the section in this section below. 14.2.14 Checking for access privileges
		72	[ChainTask] Changed the sentence in the "Remark 3." - "[Function]" as follows: When the type of the target task is basic task, the state of the task will be manipulated (the target task is shifted from SUSPENDED state to READY state) and the activation request counter will be incremented (0x1 will be added to the activation request counter). When the target task is shifted to a state other than SUSPENDED state (i.e., READY state or RUNNING state), then the state of the task will not be manipulated and only the activation request counter will be incremented. --> When the type of the target task is basic task, the state of the task will be manipulated (the target task is shifted from SUSPENDED state to READY state) and the activation request counter will be incremented (0x1 is added to the counter). When the target task has already been shifted from SUSPENDED state to READY state or RUNNING state, the state of the task will not be manipulated, the task will be queued at the end of the ready queue corresponding to the priority, and the activation request counter will be incremented.
		80 - 81	[DisableAllInterrupts] Changed the sentence in the "Scalability class 3 (SC3)" - "Remark 6." - "[Function]" as follows: When processing programs are switched, the process to enable the acceptance of interrupts will be performed (ID bit of PSW is manipulated). If a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the processing program, the process to enable the acceptance of interrupts will be performed and a hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. -->The process will differ as follows depending on the type of processing program. [Task] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		[Interrupt service routine (category 2)] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. [OS-Application-specific hook routine StartupHook_*OsApplication*] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW) and terminates the critical section. [OS-Application-specific hook routine ShutdownHook_*OsApplication*] Terminates the critical section. [Others] Correct operation is not guaranteed.
		81	[DisableAllInterrupts] Changed the sentence in the "Remark 6." - "[Function]" as follows: The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations should be performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*, and no processing is to be performed when the processing program is ErrorHook_*OsApplication*. --> The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.
		83 - 84	[SuspendAllInterrupts] Changed the sentence in the "Scalability class 3 (SC3)" - "Remark 6." - "[Function]" as follows: When processing programs are switched, the process to enable the acceptance of interrupts will be performed (ID bit of PSW is manipulated) and the disable request counter will be cleared (0x0 is set to the disable request counter). If a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the processing program, the process to enable the acceptance of interrupts will be performed, the disable request counter will be cleared, and a hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. --> The process will differ as follows depending on the type of processing program. [Task] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter. [Interrupt service routine (category 2)] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. [OS-Application-specific hook routine StartupHook_*OsApplication*] Executes the process to enable the acceptance of interrupts (manipulates the ID bit in PSW), clears the disable request counter (sets the counter to 0x0), and terminates the critical section. [OS-Application-specific hook routine ShutdownHook_*OsApplication*] Terminates the critical section. [Others] Correct operation is not guaranteed.
		84	[SuspendAllInterrupts] Changed the "Remark 6." in the "[Function]" as follows: The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations should be performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*, and no processing is to be performed when the processing program is ErrorHook_*OsApplication*. --> The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.
		85	[ResumeOSInterrupts] Added the "Remark 5." in the "[Function]" below. If this system service is issued when GetResource (with the ceiling value set to INTPRI*x*) has been issued before SuspendOSInterrupts is issued, the priority mask register (PMR) is set to the value before SuspendOSInterrupts is issued (the value indicating that the acceptance of the interrupt sources corresponding to INTPRI0 to INTPRI*x* is disabled).
		88	[SuspendedOSInterrupts] Changed the sentence in the "Scalability class 3 (SC3)" - "Remark 6." - "[Function]" as follows: When processing programs are switched, the process to enable the acceptance of interrupts will be performed (PM*n* bits of PMR are manipulated) and the disable request counter will be cleared (0x0 is set to the disable request counter). If a common hook routine (ErrorHook) or OS-Application-specific hook routine (ErrorHook_*OsApplication*) has been registered in the processing program, the process to enable the acceptance of interrupts will be performed, the disable request counter will be cleared, and a hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. --> The process will differ as follows depending on the type of processing program. [Task] Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_MISSINGEND (0x14) used as the parameter. [Interrupt service routine (category 2)] Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section. If common hook routine ErrorHook or OS-Application-specific hook routine ErrorHook_*OsApplication* has been registered, the hook routine will be called with E_OS_DISABLEDINT (0x15) used as the parameter. [OS-Application-specific hook routine StartupHook_*OsApplication*] Executes the process to enable the acceptance of interrupts (manipulates the PM*n* bits in PMR), clears the disable request counter (sets the counter to 0x0), and terminates the critical section. [OS-Application-specific hook routine ShutdownHook_*OsApplication*] Terminates the critical section. [Others] Correct operation is not guaranteed.
		88	[SuspendOSInterrupts] Changed the "Remark 6." in the "[Function]" as follows: The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations should be performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*, and no processing is to be performed when the processing program is ErrorHook_*OsApplication*. -->The AUTOSAR specifications do not specify the operations to be performed when the processing program is an OS-Application-specific hook routine. In the RV850, however, the above operations are performed when the processing program is StartupHook_*OsApplication* or ShutdownHook_*OsApplication*.
		145	[TerminateApplication] Changed the sentence in the "RESTART (0x1)" - "Remark 2." - "[Function]" as follows: Shifts the task specified in Task identifier "OsRestartTask" from SUSPENDED state to READY state. -->Shifts the target task from SUSPENDED state to READY state when Task identifier "OsRestartTask" is specified.
		146	[TerminateApplication] Added the "Remark5." in the "[Function]" below. When ErrorHook_*OsApplication* issues this system service with parameter *Application* set to "an OS-Application to which ErrorHook_*OsApplication* that issues this system service does not belong", E_OS_CALLEVEL (0x2) will be returned.
		146	[TerminateApplication] Changed the sentence in the "E_OS_ACCESS" - table - "[Return vaules]" as follows: The OS-Application to which the processing program that issued this system service belongs has no access privileges for the OS-Application specified in

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		parameter *Application* -->The OS-Application to which the processing program that issues this system service belongs is non-trusted.
		155	[GetActiveApplicationMode] Added the "Remark" in the "[Return values]" below. If an illegal value is specified in parameter *OsAppMode* for StartOS, a value not shown above may be returned.
		156 - 164	[14.4.10 Utility functions] Added the utility function in this section below. OSIllegalException_SystemRegister_ExcCode, OSIllegalException_SystemRegister_ExcPC
		156	[14.4.10 Utility functions] Added the utility function in the "Table 14.23 Utility Functions" below. OSIllegalException_SystemRegister_ExcCode, OSIllegalException_SystemRegister_ExcPC
		156	[14.4.10 Utility functions] Change the sentence in the "Remark" as follows: The AUTOSAR specifications do not prescribe utility functions InitApplicationInterrupts and _kernel_fv0_InitializeIntService. -->The AUTOSAR specifications do not prescribe utility functions InitApplicationInterrupts, _kernel_fv0_InitializeIntService, OSIllegalException_SystemRegister_ExcCode and OSIllegalException_SystemRegister_ExcPC.
		161	[OSErrorGetServiceId] Added the "Remark 2." in the "[Function]" below. When the processing program (ErrorHook or ErrorHook_*OsApplication*) that issues this utility function was called for a reason other than "abnormal end of the system service", this utility function will return an undefined value.
		164	[OSError_SystemService_*Parameter*] Added the "Remark" in the "[Function]" below. When the processing program (ErrorHook or ErrorHook_*OsApplication*) that issues this utility function was called for a reason other than "abnormal end of the system service", this utility function will return an undefined value.
		165	[A.2 Activation Method] Added the "Remark 2." in the "(1) *cf_file*" below. In the configurator, the *cf_file* is handled as an ARXML format when the extention is .arxml or .xml, as an OIL format when the extention is .oil. Only .arxml, .xml or .oil can be specified as extension of *cf_file*.
		166	[A.2 Activation Method] Added the "Remark 2." in the "(3) -o Δ *sit_file*" below. -o Δ *sit_file* can be specified multiple times as shown below. However, only the first -o Δ *sit_file* specified is treated as a valid activation option, and the others will be treated as invalid activation options.

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		Consequently, in the example below, -o Δ sit_file1.c is a valid activation option, and the other activation options (-o Δ sit_file2.c and -o Δ sit_file3.c) are ignored. C:\> Os_Configurator.exe Δ -o Δ sit_file1.c Δ -o Δ sit_file2.c Δ -o Δ sit_file3.c Δ cf_file.oil
		166	[A.2 Activation Method] Added the "Remark 2." in the "(5) -e Δ *entry_file*" below. -e Δ *entry_file* can be specified multiple times as shown below. However, only the first -e Δ *entry_file* specified is treated as a valid activation option, and the others will be treated as invalid activation options. Consequently, in the example below, -e Δ entry_file1.850 is a valid activation option, and the other activation options (-e Δ entry_file2.850 and -e Δ entry_file3.850) are ignored. C:\> Os_Configurator.exe Δ -e Δ entry_file1.850 Δ -o Δ entry_file2.850 Δ -e Δ entry_file3.850 Δ cf_file.oil
		166	[A.2 Activation Method] Added the "Remark 2." in the "(8) -I Δ *path_name*" below. This activation option can be specified multiple times (up to 255 times) as shown below. C:\> Os_Configurator.exe Δ -I Δ path_name1 Δ -I Δ path_name2 Δ -I Δ path_name3 Δ cf_file.oil
		170, 172	[A.4.1 Fatal errors] Added the error in the "Table A.1 Fatal Erros" below. E3007 E4020
		174	[A.4.2 Abort errors] Changed the message in the "F1012" - "Table A.2 Abort Errors" as follows: Too many File lines. -->Too many lines.
		176 - 177	[B.1 Overview] Added the keyword in the "(6) Keywords" below. OSTMCNT, TRACESYSTEMENTRY, TRACESYSTEMEXIT, TRACETASKSTATUS
		186	[B.4.3 OS-Application information] Changed the sentence in the "Remark 1." - "(6) SPID "*OsApplicationSPID*" as follows: This item can be specified only when FALSE is defined for Reliability "OsTrusted". --> This item can be specified only when G3M is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted".
		189	[B.4.3 OS-Application information] Changed the sentence in the "Remark 1." - "(17) Memory area identifier "*OsAppMemoryAreaNameRef*"" as follows: This item can be specified only when FALSE is defined for Reliability "OsTrusted".

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		-->
This item can be specified only when G3M is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted".
		194	[B.4.5 Event information] Added the "Remark" in this section below. In the RV850, if the event defined in this information is not defined in the Task information, fatal error E4006 will be output.
		196	[B.4.6 Interrupt service routine information] Changed the sentence in the "Remark 2." as follows: ... and the counter defined in this information is ... -->
... and the interrupt service routine defined in this information is ...
		202	[B.4.7 OS information] Changed the sentence in the "Remark 2." - "(19) Memory area identifier "*OsMemoryAreaNameRef*"" as follows: This item can be specified multiple times (up to 7 times) as shown below. -->
This item can be specified multiple times (when G3K is defined for Core identifier "OsSystemCpuCore": up to 3 times, when G3M is defined for Core identifier "OsSystemCpuCore": up to 7 times) as shown below.
		217	[C.1 Overview] Added the sentence in the "Table C.1 Memory Areas" below. .kernel_*address* Interrupt handler address table
		217	[C.2 Standard Code Area (.kernel_system)] Changed the sentence in the table as follows: ECC2, extended status, SC1, FPU not supported libecc2extsc1.a 12.0 Kbytes ECC2, extended status, SC1, FPU supported libecc2extsc1_fpu.a 12.1 Kbytes ECC2, extended status, SC3, FPU not supported libecc2extsc3.a 21.9 Kbytes ECC2, extended status, SCS, FPU supported libecc2extsc3_fpu.a 22.1 Kbytes -->
libecc2extsc1.a 12.2 Kbytes libecc2extsc1_fpu.a 12.3 Kbytes libecc2extsc3.a 22.1 Kbytes libecc2extsc3_fpu.a 22.3 Kbytes
		217	[C.2 Standard Code Area (.kernel_system)] Added the sentence in the table below. libecc2extsc3_g3k.a 21.5 Kbytes
		217	[C.3 Interface Area (.kernel_interface)] Changed the sentence in the table as follows: ECC2, extended status, SC1, FPU not supported libecc2extsc1.a ECC2, extended status, SC1, FPU supported libecc2extsc1_fpu.a ECC2, extended status, SC3, FPU not supported libecc2extsc3.a ECC2, extended status, SCS, FPU supported libecc2extsc3_fpu.a -->
libecc2extsc1.a libecc2extsc1_fpu.a libecc2extsc3.a

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		libecc2extsc3_fpu.a
		217	[C.3 Interface Area (.kernel_interface)] Added the sentence in the table below. libecc2extsc3_g3k.a 0.92 Kbytes

		218	[C.4 Constant Data Area (.kernel_const)]
			Changed the calculating formula in the "(1) Constant data area for SC1" as follows:
			
			`KERNEL_CONST =`
			`  align4 (`
			`  652`
			`  ...`
			`  + 4 * ( SystemMaxExceptionCode - 4096 )`
			`  )`
			`-->`
			`KERNEL_CONST =`
			`  align4 (`
			`  674`
			`  ...`
			`  + 4 * ( SystemMaxExceptionCode - 4095 )`
			`)

		218	[C.4 Constant Data Area (.kernel_const)] Changed the sentence in the "*AppMode_Num*" - table - "(1) Constant data area for SC1" as follows: ... of Application mode information sets -->... of Application mode information sets (except the definition of OSDEFAULTAPPMODE)

		220	[C.4 Constant Data Area (.kernel_const)]
			Changed the calculating formula in the "(2) Constant data area for SC3" as follows:
			
			`KERNEL_CONST =`
			`  align4 (`
			`  1232`
			`  ...`
			`  + 4 * ( SystemMaxExceptionCode - 4096 )`
			`  )`
			`-->`
			`KERNEL_CONST =`
			`  align4 (`
			`  1254`
			`  ...`
			`  + 4 * ( SystemMaxExceptionCode - 4095 )`
			`)

		224	[C.6 Variable Data Area (.kernel_work)]
			Changed the calculating formula in the "(1) Variable data area for SC1" as follows:
			
			`KERNEL_WORK =`
			`  align4 (`
			`  132`
			`  ...`
			`  + 28 * Task_Num`

Rev.	Date	Description	
		Page	Summary

1.01	Sep 30, 2014		```
			      ...
			  )
			 -->KERNEL_WORK =
			  align4 (
			    128
			      ...
			    + 24 * Task_Num
			      ...
			  )```

		225	[C.6 Variable Data Area (.kernel_work)]
			Changed the calculating formula in the "(2) Variable data area for SC3" as follows:
			
			``KERNEL_WORK =
			  align4 (
			    144
			      ...
			    + 36 * Task_Num
			      ...
			  )
			 -->KERNEL_WORK =
			  align4 (
			    148
			      ...
			    + 32 * Task_Num
			      ...
			  )```

		227	[C.7.1 System stack]
			Changed the calculating formula in the "(1) System stack for SC1" as follows:
			
			``SystemStack =
			  align4 (
			    60
			      ...
			    + Max (
			        ...
			      36 + StartupHookStack_Siz,
			        ...
			    )
			      ...
			  )
			 -->SystemStack =
			  align4 (
			    44
			      ...
			    + Max (
			        ...
			      20 + StartupHookStack_Siz,
			        ...
			    )
			      ...
			  )```

		227	[C.7.1 System stack] Changed the calculating formula in the "(2) System stack for SC3" as follows:

Rev.	Date	Description	
		Page	Summary

1.01	Sep 30, 2014		SystemStack =
			  align4 (
			  Max (
			  Max (
			  324,
			  264 + *ProtectionHookStack_Siz* + *ErrorHookStack_Siz*,
			  108 + *StartupHookStack_Siz* + *ErrorHookStack_Siz*
			
			  )
			  ...
			  )
			  )
			->
			SystemStack =
			  align4 (
			  Max (
			  Max (
			  320,
			  260 + *ProtectionHookStack_Siz* + *ErrorHookStack_Siz*,
			  104 + *StartupHookStack_Siz* + *ErrorHookStack_Siz*
			  )
			  ...
			  )
			  )

		230	[C.7.2 Os-Application stack]
			Changed the calculating formula in this section as follows:
			
			OsApplicationStack =
			  align4 (
			  ...
			  + MAX (
			  56 + *ErrorHookStack_Siz*,
			  ...
			  )
			  ...
			  )
			->
			OsApplicationStack =
			  align4 (
			  ...
			  + MAX (
			  104 + *ErrorHookStack_Siz*,
			  ...
			  )
			  ...
			  )

		231	[C.7.3 Task stack (basic task)]
			Changed the calculating formula in the "(1) Task stack (basic task) for SC1" as follows:
			
			TaskStack =
			  align4 (
			  160
			  ...
			  )
			->
			TaskStack =

Rev.	Date	Description	
		Page	Summary
1.01	Sep 30, 2014		align4 (164 ...)
		234	[C.7.5 Interrupt service routine stack (category 2)] Changed the calculating formula in the "(1) Interrupt service routine stack (category 2) for SC1" as follows: IsrStack = align4 (252 ...) -->IsrStack = align4 (256 ...)
		235	[C. MEMORY FOOTPRINT] Added the "C.8 Interrupt Handler Address Table (.kernel_*address*)" in this appendix.
1.02	Jun 10, 2015	17	[2.5 Writing the Linker Directive File] Changed the title of “Table 2.1” as follows. Prescribed Allocation Locations -->Object Allocation Locations Prescribed by RV850
		17	[2.5 Writing the Linker Directive File] Correct “Remark 1.” as follows. X: Write is possible W: Execution is possible -->W: Write is possible X: Execution is possible
		18	[2.6 Generation of Load Module] Added the sentence in “Remark 4.” below. When no floating-point operations are used at all (when the -fnone option is specified) in the system to be build, then in addition to the -D__WITHOSONLY__ option, the -D__NOFLOAT__ option must be specified.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	18	[2.6 Generation of Load Module] Added “Remark 5.” in this section below. When compiling the SIT file (Os_Cfg.c), it is recommended to specify the -sda=0 option. If -sda=0 is not specified, there is a possibility that the RV850 data which is supposed to be stored in the .kernel_const section and .kernel_work section as shown in table 2.1 is stored in the .sbss section and .rosdata section, respectively. In particular, when -sda=0 is not specified for an SIT file whose scalability class is SC3, the .sbss section and .rosdata section need to be allocated to an area that cannot be written by all non-trusted OS-Applications.
		19	[3.1Overview] Added the sentences in “(1)Basic tasks” below. - When the scalability class is SC1 "System stack" defined in System stack size "OsStackSize" - When the scalability class is SC3 "OS-Application stack" defined in OS-Application stack size "OsAppStackSize"
		19	[3.1Overview] Added the sentences in “(2)Extended tasks” below. The stack used while an extended task is running is the "task stack" defined in Task stack size "OsTaskStackSize".
		20	[3.1.2 Stack monitoring facilities] Added the sentences in this section below. This facility checks whether there is enough remaining amount of stacks (task stack, system stack, and OS-Application stack) needed for RV850 processing when control transfers to RV850 processing from task execution. The remaining amount of stacks will be checked at the following timing. - When starting to process a system service issued from a task - When starting pre-processing of a category 2 interrupt service routine generated during task execution
		20	[3.1.2 Stack monitoring facilities] Changed the sentence in this section below. If a stack overflow occurs inside a task, -->
If a stack overflow is detected during task execution,
		20	[3.1.2 Stack monitoring facilities] Added “Remark 2.” in this section below. The stack monitoring facilities of the RV850 cannot detect a stack overflow unless control transfers from a task to RV850 processing. If a stack overflow that could occur at a desired timing during task execution needs to be detected, use the Memory protection facility provided by non-trusted OS-Applications of scalability class SC3.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	24	[4.1.1 Stack monitoring facilities] Added the sentences in this section below. This facility checks whether there is enough remaining amount of stacks (system stack and OS-Application stack) needed for RV850 processing upon transition to RV850 processing from execution of an interrupt service routine (category 2). The remaining amount of stacks will be checked at the following timing. - When starting to process a system service issued from an interrupt service routine (category 2) - When starting pre-processing of a category 2 interrupt service routine generated during execution of an interrupt service routine (category 2)
		24	[4.1.1 Stack monitoring facilities] Changed the sentence in this section below. If a stack overflow occurs inside an interrupt service routine, -->If a stack overflow is detected during execution of an interrupt service routine (category 2),
		24	[4.1.1 Stack monitoring facilities] Added “Remark 3.” in this section below. The stack monitoring facilities of the RV850 cannot detect a stack overflow unless control transfers from an interrupt service routine (category 2) to RV850 processing. If a stack overflow that could occur at a desired timing during execution of an interrupt service routine needs to be detected, use the Memory protection facility provided by non-trusted OS-Applications of scalability class SC3.
		33	[4.5.2 Exception/interrupt safety measure process] Added basic form for coding an exception/interrupt safety measure process in the C language below. `#pragma ghs interrupt`
		33	[4.5.2 Exception/interrupt safety measure process] Added the sentence in this section below. The following points should be noted when coding an exception/interrupt safety measure process. 1) Saving/Restoring registers . . . 5) Issuing system services
		35	[5.1 Overview] Changed the sentence in this section below. The RV850 provides resource management functions as a mechanism for preventing contention over limited resources. These functions support three types of resource. -->The RV850 provides resource management facilities as a mechanism for achieving mutual exclusion. Resources are exclusively controlled using "priority ceiling protocols", and racing over resources by tasks or interrupt service routines (category 2) that use the limited number of shared resources (data, peripheral devices, common functions, etc.) or deadlocks can be prevented. The RV850 supports three types of resource.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	35	[5.1.1 Ceiling values] Moved from “5.1.2 Ceiling value” on Rev.1.01 and changed “Remark” in this section below. Interrupts that were put on hold while a resource was acquired are accepted after that resource is released. -->Interrupts that were put on hold while a resource with a ceiling value of INTPRI0 to INTPRI5 was acquired are accepted after that resource is released.
		35	[5.1.2 Scheduler resource] Moved from “5.1.1 Scheduler Resource” on Rev.1.01 and changed the sentence in this section below. The OSEK/VDX specifications have a definition of a resource, which have a identifier "RES_SCHEDULER" as a means to dynamically enable and disable the activation of the scheduler from processing programs. -->The OSEK/VDX specifications have a definition of a resource, which have a identifier "RES_SCHEDULER" and the ceiling value “29” as a means to dynamically enable and disable the activation of the scheduler from tasks.
		35	[5.1.2 Scheduler resource] Added “Remark 1.” and ”Remark 2.” in this section.
		36	[5.2 Generation of Resources] Changed the sentence in “Remark” below. The scheduler resource is generated by defining TRUE in Scheduler resource "OsUseResScheduler" and also making a definition with RES_SCHEDULER specified in Identifier "OsResource" of the Resource information. -->The OSEK/VDX specifications have a rule specifying a scheduler resource to be automatically generated by defining TRUE in Scheduler resource "OsUseResScheduler". The AUTOSAR specifications, however, have a rule specifying not to perform automatic generation. Therefore, when generating a scheduler resource in the RV850, make the following definitions in the CF file.
		42	[10.1.1 Reliability] Changed the sentence in “(1)Trusted OS-Applications” below. Note that in the RV850, since the reliability of processing programs belonging to a trusted OS-Application is ensured, they are to be operated in supervisor mode (UM bit of PSW is set to 0), and the memory protection and peripheral I/O protection functions are not applied. -->Note that in the RV850, since the reliability of processing programs belonging to a trusted OS-Application is ensured, they are to be operated in supervisor mode (UM bit of PSW is set to 0) with the system protection identifier (SPID bit of MCFG0 register) set to 0, and the memory protection and peripheral I/O protection facilities cannot be applied.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	42	[10.1.1 Reliability] Changed the sentence in “(2)Non-Trusted OS-Applications” below. Note that in the RV850, since the reliability of processing programs belonging to a non-trusted OS-Application is not ensured, they are to be operated in user mode (UM bit of PSW is set to 1), and the memory protection and peripheral I/O protection functions can be applied. -->Note that in the RV850, since the reliability of processing programs belonging to a non-trusted OS-Application is not ensured, they are to be operated in user mode (UM bit of PSW is set to 1) with the system protection identifier (SPID bit of MCFG0 register) specified by SPID "OsApplicationSPID", and the memory protection and peripheral I/O protection functions can be applied.
		43	[10.1.3 Memory protection] Added the sentence in “Remark 1.” below. The stack area used by a non-trusted OS-Application is monitored constantly regardless of the specification of Stack monitoring facilities "OsStackMonitoring"
		43	[10.1.3 Memory protection] Added “Remark 2.” in this section below. The threshold (address) of the stack pointer which is to be considered to detect an overflow in the stack area is obtained by adding the maximum value of stack usage by the system services of the RV850 to the top address of the stack area used by tasks or interrupt service routines (category 2). This is a countermeasure for the memory protection facility not operating while executing a system service of the RV850 because of the transition to supervisor mode even for a non-trusted OS-Application.
		45	[10.2 Trusted Functions] Changed the sentences in this section below. For OS-Applications whose reliability is ensured, it is possible to assign specific trusted functions to individual OS-Applications. -->For trusted OS-Applications, it is possible to assign specific trusted functions to individual OS-Applications. Trusted functions are called by issuing CallTrustedFunction from the processing program. -->Trusted functions are called by issuing CallTrustedFunction from the processing program.Since CallTrustedFunction can also be called from other non-trusted OS-Applications, trusted functions can be used when a non-trusted OS-Application is to temporarily perform processing without the protection facilities applied.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	45	[10.2 Trusted Functions] Changed the sentence in “Remark 3.” below. Trusted functions operate in supervisor mode; if CallTrustedFunction is issued from a processing program that belongs to a non-trusted OS-Application, the mode switching processing (transition from user mode to supervisor mode) is executed. -->Trusted functions operate in supervisor mode; if CallTrustedFunction is issued from a processing program that belongs to a non-trusted OS-Application, the mode switching processing (transition from user mode to supervisor mode) and system protection identifier (SPID bit of MCFG0 register) switching processing (it is set to 0 when a trusted function is being executed) are executed.
		45	[10.2 Trusted Functions] Changed the sentence in “Remark 4.” below. Since trusted functions operate in supervisor mode, the memory protection function (MPU) provided by the device cannot be applied. -->Since trusted functions operate in supervisor mode, the memory protection function cannot be applied. Also, since trusted functions operate when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection functions associated with the system protection identifier cannot be applied.
		46	[10.2.3 Inherited data of trusted functions] Added this section.
		48	[10.3 OS-Application-Specific Hook Routines] Added “Remark” in this section. Non-trusted OS-Application-specific hook routines are to be operated in user mode (UM bit of PSW is set to 1) with the system protection identifier (SPID bit of MCFG0 register) specified by SPID "OsApplicationSPID", and the memory protection and peripheral I/O protection facilities associated with the system protection identifier can be applied. Trusted OS-Application-specific hook routines are to be operated in supervisor mode (UM bit of PSW is set to 0) with the system protection identifier (SPID bit of MCFG0 register) set to 0, and the memory protection and peripheral I/O protection facilities associated with the system protection identifier cannot be applied.
		51	[11.2 Common Hook Routines] Added “Remark” in this section. Since common hook routines operate in supervisor mode, the memory protection facility cannot be applied. Also, since common hook routines operate when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection facilities associated with the system protection identifier cannot be applied.
		58	[12.3 Idle Handler] Changed the sentence in “Remark 1.” below. Since the idle handler operates in supervisor mode, the memory protection function (MPU) provided by the device cannot be applied. -->Since the idle handler operates in supervisor mode, the memory protection facility cannot be applied. Also, since the idle handler operates when the system protection identifier (SPID bit of MCFG0 register) is set to 0, peripheral I/O protection facilities associated with the system protection identifier cannot be applied.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	144	[14.4.8 OS-Application management] Changed [Function] in CallTrustedFunction below. When a trusted function is called, the value specified in parameter Function-Params is passed as the first argument to the trusted function. -->When a trusted function is called from processing of this system service, the value specified in parameter FunctionIndex is passed as the first argument to the trusted function and the value (pointer) specified in parameter FunctionParams is passed as the second argument to the trusted function. For details on the data (parameters) inherited to the trusted function, see "10.2.3 Inherited data of trusted functions"
		144	[14.4.8 OS-Application management] Changed “Remark 4.” in CallTrustedFunction below. This system service does not check the validity of parameter FunctionParams. When the validity of parameter FunctionParams needs to be checked, check it in the trusted function called by issuing this system service. -->This system service does not check the validity of the inherited data indicated by parameter FunctionParams. When the validity of the inherited data needs to be checked, check it in the trusted function called by issuing this system service.
		160	[14.4.9 OS execution management] Changed “Remark 2.” in StartOS below. Since this system service manipulates system registers, it must be issued in supervisor mode (the UM bit in PSW is 0). -->Since this system service manipulates system registers, it must be issued in supervisor mode (the UM bit in PSW is 0). Also, this system service must be issued after setting of the system protection identifier (SPID bit of MCFG0 register) (Remark 6. in “4.2 Boot Process") is complete.
		167	[14.4.10 Utility functions] Changed “Remark 2.” in _kernel_fv0_InitializeIntService below. Since this utility function manipulates system registers, it must be issued in super-visor mode (the UM bit in PSW is 0). -->Since this utility function manipulates the RV850 internal data, it must be issued in supervisor mode (the UM bit in PSW is 0).
		174	[A.2 Activation Method] Changed the method for starting the configurator from the command prompt below. OS_Configurator.exe -->Os_Configurator.exe
		195	[B.4.3 OS-Application information] Changed the title below. (3) Stack size *"OsAppStackSize"* -->(3) OS-Application Stack size *"OsAppStackSize"*

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	195	[B.4.3 OS-Application information] Changed the sentence in “(6) SPID "OsApplicationSPID"” below. Specifies the access privileges (protection through the SPID) to the I/O area. -->Specifies the access privileges (system protection identifier) to the I/O area.
		208	[B.4.7 OS information] Changed “Remark 1.” in “(4) Stack monitoring facilities "OsStackMonitoring"” below. See "3.1.2 Stack monitoring facilities" or "4.1.1 Stack monitoring facilities" for details about the stack monitoring facilities. -->See "3.1.2Stack monitoring facilities" and "4.1.1Stack monitoring facilities" for details about the stack monitoring facilities.
		209	[B.4.7 OS information] Changed the title below. (11) Stack size "*OsStackSize*" -->(11) System stack size *"OsStackSize"*
		212	[B.4.8 Resource information] Added the sentences in “Remark 1.” in “(2) Ceiling value "OsResourcePriority"” This is an optional item. When the specification is omitted, the ceiling value is automatically calculated and assigned as described in Remark 5. Except for the case of generating the scheduler resource, it is recommended to omit the specification of the ceiling value by this item, and instead specify Resource identifier "OsTaskResourceRef" to be acquired by the task or Resource identifier "OsIsrResourceRef" to be acquired by the interrupt service routine (category 2). See "5.1.1 Ceiling values" for details about the ceiling value.
		212	[B.4.8 Resource information] Changed the sentences in “Remark 3.” in “(2) Ceiling value "OsResourcePriority"” below. When RES_SCHEDULER is defined for Identifier "OsResource", OsResourcePriority should be higher than the priority specified for any Initial priority "OsTaskPriority". -->To generate the scheduler resource, RES_SCHEDULER should be defined for Identifier "OsResource", and priority 29 should be defined for OsResourcePriority.
		212	[B.4.8 Resource information] Changed the sentences in “Remark 5.” in “(2) Ceiling value "OsResourcePriority"” below. When AUTO is specified, the highest priority in all tasks and interrupt service routines for the OS-Application to which this resource belongs is found, and that priority is assigned as the ceiling value. -->When AUTO is specified, the highest priority in all tasks and interrupt service routines, which acquire the resource, defined in the CF file is obtained, and that priority is assigned as the ceiling value.

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	220	[B.4.10 Task information] Changed the title below. (5) Stack size "*OsTaskStackSize*" -->(5) Task stack size *"OsTaskStackSize"*
		237	[C.7.1 System stack] Changed the sentence of *SystemServiceStack_Max* in "(1) System stack for SC1" below. Set to 164 when extended tasks issue system services; otherwise, set to 0. -->164 (Stack size necessary for the system service issued by the extended task)
		238	[C.7.1 System stack] Added the sections below written on Rev.1.01 in "(1) System stack for SC1". (a) Task stack (basic task) for SC1 (b) Interrupt service routine stack (category 2) for SC1
		240	[C.7.2 OS-Application stack] Changed the sentence of *SystemServiceStack_Max* in this section below. Set to 164 when extended tasks issue system services; otherwise, set to 0. -->164 (Stack size necessary for the system service issued by the extended task)
		241	[C.7.2 OS-Application stack] Added the sections below written on Rev.1.01 in this section. (a) Task stack (basic task) for SC3 (b) Interrupt service routine stack (category 2) for SC3
		242	[C.7.3 Task stack (extended task)] Changed "(1) Task stack (extended task) for SC1" of *IsrContext_Siz* in this section below. Set to 124 when interrupts occur in the extended task; otherwise, set to 0. -->124 (Stack size necessary for the interrupt processing occurring during execution of the extended task)
		242	[C.7.3 Task stack (extended task)] Changed "(1) Task stack (extended task) for SC1" of *SystemServiceFrame_Siz* in this section below. Set to 8 when the extended task issues system services; otherwise set to 0. -->8 (Stack size necessary for the system service issued by the extended task)
		243	[C.7.3 Task stack (extended task)] Changed "(2) Task stack (extended task) for SC3" of *IsrContext_Siz* in this section below. Set to 128 when interrupts occur in the extended task; otherwise, set to 0. -->128 (Stack size necessary for the interrupt processing occurring during execution of the extended task)

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	243	[C.7.3 Task stack (extended task)] Changed "(2) Task stack (extended task) for SC3" of *SystemServiceFrame_Siz* in this section below. Set to 12 when the extended task issues system services; otherwise set to 0. -->12 (Stack size necessary for the system service issued by the extended task)
		21	[3.1.4 Processing in tasks] Added "Remark 2." in "(2) Saving/Restoring FPSR".
		28	[4.3.1 Processing in interrupt service routines] Added "Remark" in "(1) Category 1".
		28	[4.3.1 Processing in interrupt service routines] Added "Remark 2." in "(2) Category 2".
		39	[8.2.1 Processing in alarm callbacks] Added "Remark" in "(2) Saving/Restoring FPSR".
		45	[10.2.1 Processing in trusted functions] Added "Remark 2." in "(2) Saving/Restoring FPSR".
		49	[10.3.1 Processing in OS-Application-specific hook routines] Added "Remark" in "(2) Saving/Restoring FPSR".
		54	[11.2.1 Processing in common hook routines] Added "Remark" in "(2) Saving/Restoring FPSR".
		226	[C.2 Standard Code Area (.kernel_system)] Changed the memory size value used by the kernel libraries below due to version upgrade (Modification of kernel libraries) to V2.01.01. libecc2extsc1_fpu.a 12.3Kbytes --> 12.4Kbytes libecc2extsc3_g3k.a 21.5Kbytes --> 21.6Kbytes
		227	[C.4 Constant Data Area (.kernel_const)] Changed the fixed value of formula for estimating the memory size in "(1) Constant data area for SC1 (.kernel_const)" below due to version upgrade (Modification of kernel libraries) to V2.01.01. 674 --> 673
		229	[C.4 Constant Data Area (.kernel_const)] Changed the fixed value of formula for estimating the memory size in "(2) Constant data area for SC3 (.kernel_const)" below due to version upgrade (Modification of kernel libraries) to V2.01.01. 1254 --> 1253
		239	[C.7.1 System stack] Changed the fixed value of formula for estimating the memory size in "(2) System stack for SC3" below due to version upgrade (Modification of kernel libraries) to V2.01.01. 260 --> 264

Rev.	Date	Description	
		Page	Summary
1.02	Jun 10, 2015	-	[1.5 Folder Structure] Changed the name of RV850 installation destination folder below. <rx_root> --> <rv_root>
		-	Changed the description of system register MCFG0.SPID below. System protection number --> System protection identifier
		220	[B.4.10 Task information] Changed the sentences in “Remark 2.” in “(5) Task stack size "OsTaskStackSize"” below. See "C.7.3 Task stack (basic task)" or "C.7.4 Task stack (extended task)" for details about the size specified in this item. -->` See "C.7.3Task stack (extended task)" for details about the size specified in this item.
		244	[C.8 Interrupt Handler Address Table (.kernel_address)] Changed the calculating formula in this section below. `IntHdrAdrTbl = 16 * InterruptSource_Num --> IntHdrAdrTbl = 4 * (SystemMaxExceptionCode - 4095)`
1.03	Dec 17, 2015	10	[1.4 Execution Environment] Changed information of the target devices in “(1)Devices” below. RH850 family (G3K core, G3M core) -->` RH850 family (G3K core, G3M core, G3KH core, G3MH core)
		10	[1.4 Execution Environment] Changed the sentences in “(2)Peripheral controllers” in “Remark 3.” below. ...scalability class SC3 (only in G3M core) -->` ...scalability class SC3 (only in G3M core, G3KH core, G3MH core)
		10	[1.4 Execution Environment] Changed the sentences in “Table 1.1OS Reserved Resources Occupied by RV850” in “SPID bit of machine configuration (MCFG0)” below. SC3 (only in G3M core) -->` SC3 (only in G3M core, G3KH core, G3MH core)

Rev.	Date	Description	
		Page	Summary
1.03	Dec 17, 2015	11	[1.5.1Object release version] Changed the sentences in “(9)” below. libecc2extsc3.a: ECC2, extended status, SC3, FPU not supported (only in G3M core) libecc2extsc3_fpu.a: ECC2, extended status, SC3, FPU supported (only in G3M core) - ->> libecc2extsc3.a: ECC2, extended status, SC3, FPU not supported (only in G3M, G3KH, G3MH core) libecc2extsc3_fpu.a: ECC2, extended status, SC3, FPU supported (only in G3M, G3KH, G3MH core)
		24	[4.2 Boot Process] Changed the sentences in “(2)” below. Interrupt function registers - ->> Interrupt function registers (only in G3M core, G3KH core)
		24	[4.2 Boot Process] Changed the sentences in “(4)” below. The safety function associated with the SPID bit (system protection identifier) of the machine configuration register (MCFG0) (only in G3M core) - ->> The safety function associated with the SPID bit (system protection identifier) of the machine configuration register (MCFG0) (only in G3M core, G3KH core, G3MH core)
		185	[B.1 Overview] Added new keywords in “(6) Keywords” below G3KH, G3MH
		195	[B.4.3 OS-Application information] Changed the sentences in “(6) SPID "OsApplicationSPID"” in ”Remark 1.” below this item can be specified when G3M is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted". - ->> this item can be specified when G3M or G3KH or G3MH is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted".
		198	[B.4.3 OS-Application information] Changed the sentences in “(17) Memory area identifier "OsAppMemoryAreaNameRef"” in “Remark 1.” below This item can be specified when G3M is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted". - ->> tThis item can be specified when G3M or G3KH or G3MH is defined for Core identifier "OsSystemCpuCore", and FALSE is defined for Reliability "OsTrusted".

Rev.	Date	Description	
		Page	Summary
1.03	Dec 17, 2015	209	[B.4.7 OS information] Changed the sentences in “(12) FPSR default value "OsDefaultFPSRValue"” in below [If omitted:] Processing is performed assuming that 0x20000 is specified. - ->[If omitted:] Processing is performed assuming that reset value of the target device is specified.
		211	[B.4.7 OS information] Changed the sentences in “(19) Memory area identifier "OsMemoryAreaNameRef"” in below This item can be specified multiple times (when G3K is defined for Core identifier "OsSystemCpuCore": up to 3 times, when G3M is defined for Core identifier "OsSystemCpuCore": up to 7 times) as shown below. - ->This item can be specified multiple times (when G3K is defined for Core identifier "OsSystemCpuCore": up to 3 times, when G3M or G3KH or G3MH is defined for Core identifier "OsSystemCpuCore": up to 7 times) as shown below.
		223	[B.4.11 System information] Changed the sentences in “(3) Core identifier "OsSystemCpuCore"” in below Only G3K or G3M can be specified as *OsSystemCpuCore*. - ->Only G3K or G3M or G3KH or G3MH can be specified as *OsSystemCpuCore*.

RV850 Real-Time Operating System User's Manual: Functionality

Publication Date: Rev.1.00 Apr 25, 2014
Rev.1.03 Dec 17, 2015
Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

RV850

Renesas Electronics Corporation

R20UT2768EJ0103