User's Manual

CS+ Code Generator Tool

Integrated Development Environment

User's Manual: RX API Reference

Target Device

RX Family

Target Tool

CS+ V4.01.00

Renesas Electronics
www.renesas.com

Rev.1.30 Oct 2016

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

This manual describes the role of the CS+ integrated development environment for developing applications and systems for RX family, and provides an outline of its features.
CS+ is an integrated development environment (IDE) for RX family, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.
By providing an integrated environment, it is possible to perform all development using just this product, without the need to use many different tools separately.

Readers

This manual is intended for users who wish to understand the functions of the CS+ and design software and hardware application systems.

Purpose

This manual is intended to give users an understanding of the functions of the CS+ to use for reference in developing the hardware or software of systems using these devices.
We aim to help their system development including their hardware and software.

Organization

This manual can be broadly divided into the following units.

1.GENERAL
2.OUTPUT FILES
3.API FUNCTIONS

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{XXX}$ (overscore over pin or signal name)
Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Decimal ... XXXX Hexadecimal ... 0xXXXX

TABLE OF CONTENTS

1. GENERAL

Code Generator Tool is a software tool that automatically generates device drivers.
This manual explains about .
This manual gives Output files and API functions.

1.1 Overview

Code Generator tool enables you to output the pin assignment of the microcontroller (device pin list and device top view), and the source code (device driver programs, C source files and header files) necessary to control the peripheral functions (clock generator, port functions, etc.) provided by the microcontroller by configuring various information using the GUI.

1.2 Features

The Code Generator tool has the following features.

- Code generating function
  The Code Generator can output not only device driver programs in accordance with the information configured using the GUI, but also a build environment such as sample programs containing main functions and link directive files.

- Reporting function
  You can output configured information using the Pin Configurator/Code Generator as files in various formats for use as design documents.

- Renaming function
  The user can change default names assigned to the files output by the Code Generator and the API functions contained in the source code.

- User code protective function
  The user can add user's original source code to each API function.When user generated the device driver programs again by the Code Generator, user's source code within this comment is protected.

  [Comment for user source code descriptions]
  /* Start user code. Do not edit comment generated here */
  /* End user code. Do not edit comment generated here */

2. OUTPUT FILES

This appendix describes the files output by the Code Generator.

2.1 Description

Below is a list of output file files by the Code Generator.

Table 2.1 Output File List

Peripheral Function	File Name	API Function Name	out put (*1)
Common	r_cg_dbsct.c	-	-
	r_cg_hardware_setup.c	HardwareSetup R_Systeminit	A A
	r_cg_intprg.c	r_undefined_exception r_privileged_exception r_floatingpoint_exception r_access_exception r_nmi_exception r_brk_exception r_reserved_exception	A A A A A A A
	r_cg_main.c	main R_MAIN_UserInit	A A
	r_cg_resetprg.c	PowerON_Reset PowerON_Reset_PC	A A
	r_cg_sbrk.c	-	-
	r_cg_vecttbl.c	-	-
	r_cg_macrodriver.h	-	-
	r_cg_sbrk.h	-	-
	r_cg_stacksct.h	-	-
	r_cg_userdefine.h	-	-
	r_cg_vect.h	-	-
Clock generation circuit	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode	A M
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_oscillation_stop_interrupt r_cgc_oscillation_stop_nmi_interrupt	M A A
	r_cg_cgc.h	-	-
Voltage detection circuit (LVDA)	r_cg_lvd.c	R_LVDn_Create R_LVDn_Start R_LVDn_Stop	A A A
	r_cg_lvd_user.c	R_LVDn_Create_UserInit r_lvd_lvdn_interrupt	M A
	r_cg_lvd.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
Clock frequency accuracy measurement circuit (CAC)	r_cg_cac.c	R_CAC_Create R_CAC_Start R_CAC_Stop	A A A
	r_cg_cac_user.c	R_CAC_Create_UserInit r_cac_mendf_interrupt r_cac_ferrf_interrupt r_cac_ovff_interrupt	M A A A
	r_cg_cac.h	-	-
Low power consumption	r_cg_lpc.c	R_LPC_Create R_LPC_AllModuleClockStop R_LPC_ChangeSleepModeRetrunClock R_LPC_Sleep R_LPC_DeepSleep R_LPC_DeepSoftwareStandby R_LPC_SoftwareStandby R_LPC_ChangeOperationPowerControl	A A A A A A A A
	r_cg_lpc_user.c	R_LPC_Create_UserInit	M
	r_cg_lpc.h	-	-
Interrupt controller (ICU)	r_cg_icu.c	R_ICU_Create R_ICU_IRQn_Start R_ICU_IRQn_Stop R_ICU_Software_Start R_ICU_Software2_Start R_ICU_Software_Stop R_ICU_Software2_Stop R_ICU_SoftwareInterrupt_Generate R_ICU_SoftwareInterrupt2_Generate	A A A A A A A A A
	r_cg_icu_user.c	R_ICU_Create_UserInit r_icu_irqn_interrupt r_icu_software_interrupt r_icu_software2_interrupt r_icu_nmi_interrupt	M A A A A
	r_cg_icu.h	-	-
Buses	r_cg_bsc.c	R_BSC_Create R_BSC_Error_Monitoring_Start R_BSC_Error_Monitoring_Stop R_BSC_InitializeSDRAM	A A A A
	r_cg_bsc_user.c	R_BSC_Create_UserInit r_bsc_buserr_interrupt	M A
	r_cg_bsc.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
DMA Controller(DMAC)	r_cg_dmac.c	R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTrigger R_DMACn_Clear_SoftwareTrigger	A A A A A
	r_cg_dmac_user.c	r_dmac_dmacni_interrupt r_dmacn_callback_transfer_end r_dmacn_callback_transfer_escape_end R_DMAC_Create_UserInit	A A A M
	r_cg_dmac.h	-	-
Data transfer controller (DTC)	r_cg_dtc.c	R_DTC_Create R_DTCm_Start R_DTCm_Stop	A A A
	r_cg_dtc_user.c	R_DTC_Create_UserInit	M
	r_cg_dtc.h	-	-
Event link controller (ELC)	r_cg_elc.c	R_ELC_Create R_ELC_Start R_ELC_Stop R_ELC_GenerateSoftwareEvent R_ELC_Set_PortBuffern R_ELC_Get_PortBuffern	A A A A A A
	r_cg_elc_user.c	R_ELC_Create_UserInit r_elc_elsrni_interrupt	M A
	r_cg_elc.h	-	-
I/O ports	r_cg_port.c	R_PORT_Create	A
	r_cg_port_user.c	R_PORT_Create_UserInit	M
	r_cg_port.h	-	-
Multi-function timer pulse unit 2 (MTU2)	r_cg_mtu2.c	R_MTU2_Create R_MTU2_Cn_Start R_MTU2_Cn_Stop	A A A
	r_cg_mtu2_user.c	R_MTU2_Create_UserInit r_mtu2_tgimn_interrupt r_mtu2_cj_tgimn_interrupt r_mtu2_tcivn_interrupt r_mtu2_cj_tcivn_interrupt r_mtu2_tciun_interrupt	M A A A A A
	r_cg_mtu2.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
Multi-function timer pulse unit 3 (MTU3)	r_cg_mtu3.c	R_MTU3_Create R_MTU3_Cn_Start R_MTU3_Cn_Stop	A A A
	r_cg_mtu3_user.c	R_MTU3_Create_UserInit r_mtu3_tgimn_interrupt r_mtu3_cj_tgimn_interrupt r_mtu3_tcivn_interrupt r_mtu3_cj_tcivn_interrupt r_mtu3_tciun_interrupt	M A A A A A
	r_cg_mtu3.h	-	-
Port output enable 2 (POE2)	r_cg_poe2.c	R_POE2_Create R_POE2_Start R_POE2_Stop R_POE2_Set_HiZ_MTUn R_POE2_Clear_HiZ_MTUn	A A A A A
	r_cg_poe2_user.c	R_POE2_Create_UserInit r_poe2_oein_interrupt	M A
	r_cg_poe2.h	-	-
Port output enable 3 (POE3)	r_cg_poe3.c	R_POE3_Create R_POE3_Start R_POE3_Stop R_POE3_Set_HiZ_MTUn R_POE3_Clear_HiZ_MTUn R_POE3_Set_HiZ_GPTn R_POE3_Clear_HiZ_GPTn	A A A A A A A
	r_cg_poe3_user.c	R_POE3_Create_UserInit r_poe3_oein_interrupt	M A
	r_cg_poe3.h	-	-
General PWM timer (GPT)	r_cg_gpt.c	R_GPT_Create R_GPTn_Start R_GPTn_Stop R_GPTn_HardwareStart R_GPTn_HardwareStop	A A A A A
	r_cg_gpt_user.c	R_GPT_Create_UserInit r_gpt_gtcimn_interrupt r_gpt_gtcivn_interrupt r_gpt_gtciun_interrupt r_gpt_gdten_interrupt r_gpt_etgip_interrupt r_gpt_etgin_interrupt	M A A A A A A
	r_cg_gpt.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
16-bit timer pulse unit (TPU)	r_cg_tpu.c	R_TPU_Create R_TPUn_Start R_TPUn_Stop	A A A
	r_cg_tpu_user.c	R_TPU_Create_UserInit r_tpu_tginm_interrupt r_tpu_tcinv_interrupt r_tpu_tcinu_interrupt	M A A A
	r_cg_tpu.h	-	-
8-bit timer (TMR)	r_cg_tmr.c	R_TMR_Create R_TMRn_Start R_TMRn_Stop	A A A
	r_cg_tmr_user.c	R_TMR_Create_UserInit r_tmr_cmimn_interrupt r_tmr_ovin_interrupt	M A A
	r_cg_tmr.h	-	-
Programmable pulse generator (PPG)	r_cg_ppg.c	R_PPG_Create	A
	r_cg_ppg_user.c	R_PPG_Create_UserInit	M
	r_cg_ppg.h	-	-
Compare match timer (CMT)	r_cg_cmt.c	R_CMTn_Create R_CMTn_Start R_CMTn_Stop	A A A
	r_cg_cmt_user.c	R_CMTn_Create_UserInit r_cmt_cmin_interrupt	M A
	r_cg_cmt.h	-	-
Compare match timer W (CMTW)	r_cg_cmtw.c	R_CMTWn_Create R_CMTWn_Start R_CMTWn_Stop	A A A
	r_cg_cmtw_user.c	R_CMTWn_Create_UserInit r_cmtw_cmwin_interrupt r_cmtw_icmin_interrupt r_cmtw_ocmin_interrupt	M A A A
	r_cg_cmtw.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
Realtime clock (RTC)	r_cg_rtc.c	R_RTC_Create R_RTC_Set_CalendarAlarm R_RTC_Set_BinaryAlarm R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_CarryInterruptOn R_RTC_Set_CarryInterruptOff R_RTC_Set_RTCOUTOn R_RTC_Set_RTCOUTOff R_RTC_Start R_RTC_Stop R_RTC_Restart R_RTC_Set_CalendarCounterValue R_RTC_Get_CalendarCounterValue R_RTC_Set_BinaryCounterValue R_RTC_Get_BinaryCounterValue R_RTC_Get_CalendarTimeCaptureValuen R_RTC_Get_BinaryTimeCaptureValuen	A A A A A A A A A A A A A A A A A A
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_alm_interrupt r_rtc_prd_interrupt r_rtc_cup_interrupt	M A A A
	r_cg_rtc.h	-	-
Watchdog timer (WDT)	r_cg_wdt.c	R_WDT_Create R_WDT_Restart	A A
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_nmi_interrupt r_wdt_wuni_interrupt	M A A
	r_cg_wdt.h	-	-
Independent watchdog timer (IWDT)	r_cg_iwdt.c	R_IWDT_Create R_IWDT_Restart	A A
	r_cg_iwdt_user.c	R_IWDT_Create_UserInit r_iwdt_nmi_interrupt r_iwdt_iwuni_interrupt	M A A
	r_cg_iwdt.h	-	-

Peripheral Function	File Name	API Function Name	out put (*1)
Serial communications interface (SCI)	r_cg_sci.c	R_SCIn_Create R_SCIn_Start R_SCIn_Stop R_SCIn_Serial_Send R_SCIn_Serial_Receive R_SCIn_Serial_Multiprocessor_Send R_SCIn_Serial_Multiprocessor_Receive R_SCIn_Serial_Send_Receive R_SCIn_SmartCard_Send R_SCIn_SmartCard_Receive R_SCIn_IIC_Master_Send R_SCIn_IIC_Master_Receive R_SCIn_SPI_Master_Send R_SCIn_SPI_Master_Send_Receive R_SCIn_SPI_Slave_Send R_SCIn_SPI_Slave_Send_Receive R_SCIn_IIC_StartCondition R_SCIn_IIC_StopCondition	A A A A A A A A A A A A A A A A A A
	r_cg_sci_user.c	R_SCIn_Create_UserInit r_scin_transmitend_interrupt r_scin_transmit_interrupt r_scin_receive_interrupt r_scin_receiveerror_interrupt r_scin_callback_transmitend r_scin_callback_receiveend r_scin_callback_receiveerror	M A A A A A A A
	r_cg_sci.h	-	-
FIFO embedded serial communica-tions interface (SCIFA)	r_cg_scifa.c	R_SCIFAn_Create R_SCIFAn_Start R_SCIFAn_Stop R_SCIFAn_Serial_Send R_SCIFAn_Serial_Receive R_SCIFAn_Serial_Send_Receive	A A A A A A
	r_cg_scifa_user.c	R_SCIFAn_Create_UserInit r_scifan_teif_interrupt r_scifan_txif_interrupt r_scifan_rxif_interrupt r_scifan_erif_interrupt r_scifan_brif_interrupt r_scifan_drif_interrupt r_scifan_callback_transmitend r_scifan_callback_receiveend r_scifan_callback_error	M A A A A A A A A A
	r_cg_scifa.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
I^{2}C bus interface (RIIC)	r_cg_riic.c	R_RIICn_Create R_RIICn_Start R_RIICn_Stop R_RIICn_Master_Send R_RIICn_Master_Receive R_RIICn_Slave_Send R_RIICn_Slave_Receive R_RIICn_StartCondition R_RIICn_StopCondition	A A A A A A A A A
	r_cg_riic_user.c	R_RIICn_Create_UserInit r_riicn_error_interrupt r_riicn_receive_interrupt r_riicn_transmit_interrupt r_riicn_transmitend_interrupt r_riicn_callback_receiveerror r_riicn_callback_transmitend r_riicn_callback_receiveend	M A A A A A A A
	r_cg_riic.h	-	-
Serial peripheral interface (RSPI)	r_cg_rspi.c	R_RSPIn_Create R_RSPIn_Start R_RSPIn_Stop R_RSPIn_Send R_RSPIn_Send_Receive	A A A A A
	r_cg_rspi_user.c	R_RSPIn_Create_UserInit r_rspin_receive_interrupt r_rspin_transmit_interrupt r_rspin_error_interrupt r_rspin_idle_interrupt r_rspin_callback_receiveend r_rspin_callback_error r_rspin_callback_transmitend	M A A A A A A A
	r_cg_rspi.h	-	-
CRC calculator (CRC)	r_cg_crc.c	R_CRC_SetCRC8 R_CRC_SetCRC16 R_CRC_SetCCITT R_CRC_SetCRC32 R_CRC_SetCRC32C R_CRC_Input_Data R_CRC_Get_Result	A A A A A A A
	r_cg_crc.h	-	-
12-bit A/D converter (S12AD)	r_cg_s12ad.c	R_S12ADn_Create R_S12ADn_Start R_S12ADn_Stop R_S12ADn_Get_ValueResult R_S12ADn_Set_CompareValue	A A A A A
	r_cg_s12ad_user.c	R_S12ADn_Create_UserInit r_s12adn_interrupt r_s12adn_groupb_interrupt r_s12adn_compare_interrupt	M A A A
	r_cg_s12ad.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
D/A converter (DA)	r_cg_da.c	R_DA_Create R_DAm_Start R_DAm_Stop R_DAm_Set_ConversionValue	A A A A
	r_cg_da_user.c	R_DA_Create_UserInit	M
	r_cg_da.h	-	-
12-bit converter (R12DA)	r_cg_r12da.c	R_R12DA_Create R_R12DAn_Start R_R12DAn_Stop R_R12DAn_Set_ConversionValue R_R12DA_sync_Start R_R12DA_sync_Stop	A A A A A A
	r_cg_r12da_user.c	R_DA_Create_UserInit	M
	r_cg_r12da.h	-	-
Comparator B (CMPB)	r_cg_cmpb.c	R_CMPB_Create R_CMPBn_Start R_CMPBn_Stop	A A A
	r_cg_cmpb_user.c	R_CMPB_Create_UserInit r_cmpb_cmpbn_interrupt	M A
	r_cg_cmpb.h	-	-
Data operation circuit (DOC)	r_cg_doc.c	R_DOC_Create R_DOC_SetMode R_DOC_WriteData R_DOC_GetResult R_DOC_ClearFlag	A A A A A
	r_cg_doc_user.c	R_DOC_Create_UserInit r_doc_dopcf_interrupt	M A
	r_cg_doc.h	-	
Low power timer (LPT)	r_cg_lpt.c	R_LPT_Create R_LPT_Start R_LPT_Stop	A A A
	r_cg_lpt_user.c	R_LPT_Create_UserInit	M
	r_cg_lpt.h	-	-
Comparator C (CMPC)	r_cg_cmpc.c	R_CMPC_Create R_CMPCn_Start R_CMPCn_Stop	A A A
	r_cg_cmpc_user.c	R_CMPC_Create_UserInit r_cmpc_cmpcn_interrupt	M A
	r_cg_cmpc.h	-	-

Peripheral Function	File Name	API Function Name	output (*1)
LCD controller / driver (LCD)	r_cg_cld.c	R_LCD_Create R_LCD_Start R_LCD_Stop R_LCD_Voltage_On R_LCD_Voltage_Off	A A A A A
	r_cg_lcd_user.c	R_LCD_Create_UserInit	M
	r_cg_lcd.h	-	-

*1 In case of [API output control] setting are default ([Output all API functions according to the setting]).
A : Output by settings on each peripheral functions panel automatically.
M : Output by the file used setting in API property.

3. API FUNCTIONS

This appendix describes the API functions output by the Code Generator.

3.1 Overview

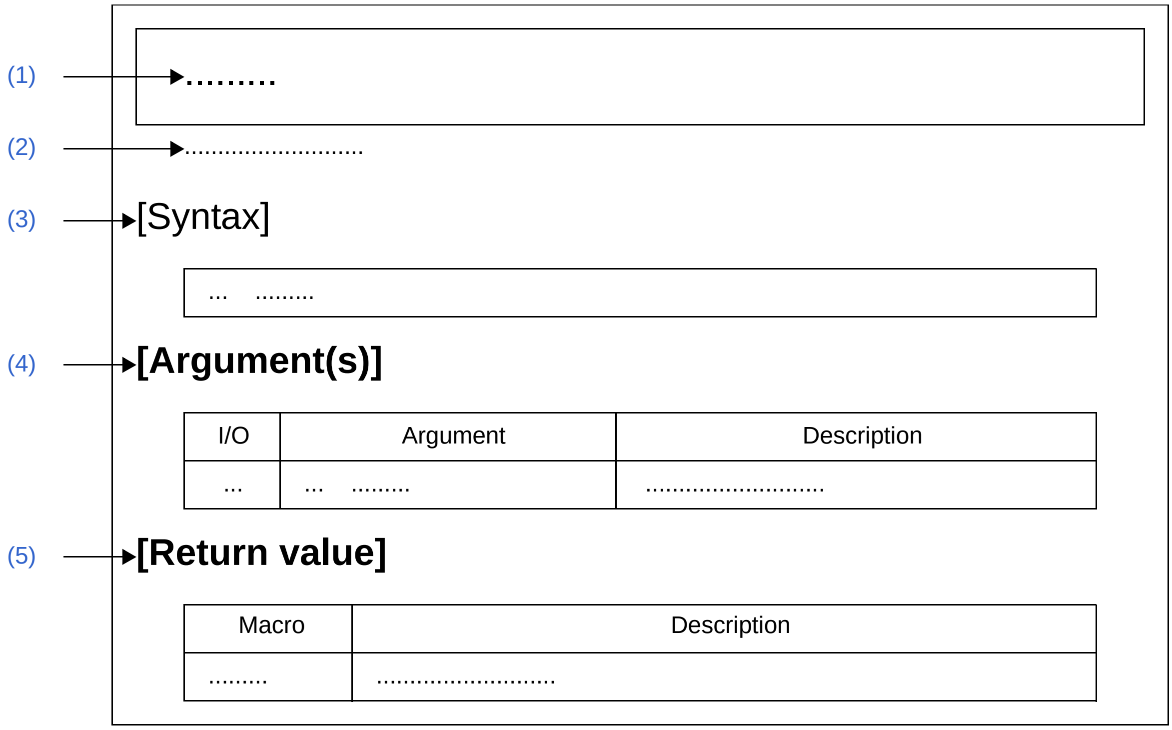
Below are the naming conventions for API functions output by the Code Generator.

- Macro names are in ALL CAPS.
  The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.
- Local variable names are in all lower case.
- Global variable names start with a "g" and use Camel Case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

3.2 Function Reference

This section describes the API functions output by the Code Generator, using the following notation format.

Figure 3.1 Notation Format of API Functions



(1) Name
Indicates the name of the API function.

(2) Outline
Outlines the functions of the API function.

(3) [Syntax]
Indicates the format to be used when describing an API function to be called in C language.

(4) [Argument(s)]
API function arguments are explained in the following format.

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

(a) I/O
Argument classification
I ... Input argument
O ... Output argument

(b) Argument
Argument data type

(c) Description
Description of argument

(5) [Return value]
API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro
Macro of return value

(b) Description
Description of return value

3.2.1 Common

Below is a list of API functions output by the Code Generator for common use.
Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).

Table 3.1 API Functions: [Common]

API Function Name	Function
r_undefined_exception	Performs processing in response to the undefined instruction exception.
PowerON_Reset	Performs processing in response to the reset.
PowerON_Reset_PC	Performs processing in response to the reset
r_privileged_exception	Performs processing in response to the privileged instruction exception.
r_floatingpoint_exception	Performs processing in response to the floating-point exception.
r_access_exception	Performs processing in response to the access exception.
r_nmi_exception	Performs processing in response to the non-maskable interrupt.
r_brk_exception	Performs processing in response to the unconditional trap.
r_reserved_exception	Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).
HardwareSetup	Performs initialization necessary to control the various hardwares.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.
r_icu_group_n_interrupt	Performs processing in response to the group interrupt.

r_undefined_exception

Performs processing in response to the undefined instruction exception.

Remark This API function is called to run interrupt processing in response to an undefined instruction exception occurred when detecting the undefined instruction (unimplemented instruction) execution.

[Syntax]

```
void    r_undefined_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

PowerON_Reset

Performs processing in response to the reset.

Remark This API function is called to run interrupt processing for an internal reset by the power-on reset circuit.

[Syntax]

```
void    PowerON_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

PowerON_Reset_PC

Performs processing in response to the reset.

Remark This API function is called to run interrupt processing for an internal reset by the power-on reset circuit.

[Syntax]

```
void    PowerON_Reset_PC ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_privileged_exception

Performs processing in response to the privileged instruction exception.

Remark This API function is called to run interrupt processing in response to a privilegedundefined instruction exception occurred when detecting the execution of a privileged instruction in user mode.

[Syntax]

```
void    r_privileged_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_floatingpoint_exception

Performs processing in response to the floating-point exception.

Remark This API function is called to run interrupt processing in response to a floating-point exception occurred when detecting the five exception events (overflow, underflow, inexact, division-by-zero, and invalid operation) specified in the IEEE754 standard and another floating-point exception that is generated on detection of unimplemented processing.

[Syntax]

```
void    r_floatingpoint_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_access_exception

Performs processing in response to the access exception.

Remark This API function is called to run interrupt processing in response to an access exception occurred when detecting the memory-protection error, and data memory protection error.

[Syntax]

```
void    r_access_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_nmi_exception

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt.

[Syntax]

```
void    r_nmi_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_brk_exception

Performs processing in response to the unconditional trap.

Remark This API function is called to run interrupt processing for an unconditional trap.

[Syntax]

```
void    r_brk_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_reserved_exception

Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).

Remark This API function is called to run interrupt processing for the exceptions other than undefined instruction exception, reset, non-maskable interrupt, and unconditional trap.

[Syntax]

```
void    r_reserved_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

HardwareSetup

Performs initialization necessary to control the various hardwares.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    HardwareSetup ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_Systeminit

Performs initialization necessary to control the various peripheral functions.

Remark This API function is called as the HardwareSetup callback routine.

[Syntax]

```
void    R_Systeminit ( void );
```

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    main ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAIN_UserInit

Performs user-defined initialization.

Remark This API function is called as the main callback routine.

[Syntax]

```
void    R_MAIN_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_group_*n*_interrupt

Performs processing in response to the group interrupts.

[Syntax]

```
void    r_icu_group_n_interrupt ( void );
```

Remark *n* is the group interrupt number.

[Argument(s)]

None.

[Return value]

None.

3.2.2 Clock generation circuit

Below is a list of API functions output by the Code Generator for clock generation circuit use.

Table 3.2 API Functions: [Clock Generation Circuit]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generation circuit.
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generation circuit.
r_cgc_oscillation_stop_interrupt	Performs processing in response to the osillation stop detection interrupt.
r_cgc_oscillation_stop_nmi_interrupt	Performs processing in response to the oscillation stop detection NMI.
R_CGC_Set_ClockMode	Sets the clock source.

R_CGC_Create

Performs initialization required to control the clock generation circuit.

[Syntax]

```
void    R_CGC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

Performs user-defined initialization relating to the clock generation circuit.

Remark This API function is called as the R_CGC_Create callback routine.

[Syntax]

```
void    R_CGC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_oscillation_stop_interrupt

Performs processing in response to the oscillation stop detection interrupt.

Remark This API function is called to run interrupt processing for the oscillation stop detection interrupt, which is generated when the clock generation circuit detects oscillation by the main clock having stopped.

[Syntax]

```
static  void    r_cgc_oscillation_stop_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_oscillation_stop_nmi_interrupt

Performs processing in response to the oscillation stop detection NMI.

[Syntax]

```
static  void    r_cgc_oscillation_stop_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Sets the clock source.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_cgc.h"
MD_STATUS   R_CGC_Set_ClockMode ( clock_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	`clock_mode_t mode;`	Clock source type MAINCLK: Main clock oscillator SUBCLK: Sub-clock oscillator PLLCLK: PLL circuit HOCO: High-speed on-chip oscillator LOCO: Low-speed on-chip oscillator

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ARGERROR	Invalid argument *mode* specification

3.2.3 Voltage detection circuit (LVDA)

Below is a list of API functions output by the Code Generator for voltage detection circuit use.

Table 3.3 API Functions: [Voltage Detection Circuit]

API Function Name	Function
R_LVDn_Create	Performs initialization necessary to control the voltage detection circuit.
R_LVDn_Create_UserInit	Performs user-defined initialization relating to the voltage detection circuit.
r_lvd_lvdn_interrupt	Performs processing in response to the voltage monitoring *n* interrupt.
R_LVDn_Start	Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).
R_LVDn_Stop	Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

R_LVD*n*_Create

Performs initialization necessary to control the voltage detection circuit.

[Syntax]

```
void    R_LVDn_Create ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Create_UserInit

Performs user-defined initialization relating to the voltage detection circuit.

Remark This API function is called as the R_LVDn_Create callback routine.

[Syntax]

```
void    R_LVDn_Create_UserInit ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

r_lvd_lvd*n*_interrupt

Performs processing in response to the voltage monitoring *n* interrupt.

Remark This API function is called to run interrupt processing for the voltage monitoring *n* interrupt, which is generated when the voltage detection circuit detects the voltage being dropped.

[Syntax]

```
static  void    r_lvd_lvdn_interrupt ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Start

Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Start ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Stop

Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Stop ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

3.2.4 Clock frequency accuracy measurement circuit (CAC)

Below is a list of API functions output by the Code Generator for clock frequency accuracy measurement circuit use.

Table 3.4 API Functions: [Clock Frequency Accuracy Measurement Circuit]

API Function Name	Function
R_CAC_Create	Performs initialization necessary to control the clock frequency accuracy measurement circuit.
R_CAC_Create_UserInit	Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.
r_cac_mendf_interrupt	Performs processing in response to the measurement end interrupt.
r_cac_ferrf_interrupt	Performs processing in response to the frequency error interrupt.
r_cac_ovff_interrupt	Performs processing in response to the overflow interrupt.
R_CAC_Start	Starts measurement of the accuracy of the clock frequency.
R_CAC_Stop	Ends measurement of the accuracy of the clock frequency.

R_CAC_Create

Performs initialization necessary to control the clock frequency accuracy measurement circuit.

[Syntax]

```
void    R_CAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Create_UserInit

Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.

Remark This API function is called as the R_CAC_Create callback routine.

[Syntax]

```
void    R_CAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_mendf_interrupt

Performs processing in response to the measurement end interrupt.

Remark This API function is called to run interrupt processing for the measurement end interrupt, which is generated when the clock frequency accuracy measurement circuit detects the valid edge of the reference signal.

[Syntax]

```
static  void    r_cac_mendf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ferrf_interrupt

Performs processing in response to the frequency error interrupt.

Remark This API function is called to run interrupt processing for the frequency error interrupt, which is generated when the clock frequency is not in the allowed range (from the minimum to the maximum value).

[Syntax]

```
static  void    r_cac_ferrf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ovff_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated when the counter overflows.

[Syntax]

```
static  void    r_cac_ovff_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Start

Starts measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Stop

Ends measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.5 Low power consumption

Below is a list of API functions output by the Code Generator for low power consumption use.

Table 3.5 API Functions: [Low Power Consumption]

API Function Name	Function
R_LPC_Create	Performs initialization required to control the low power consumption.
R_LPC_Create_UserInit	Performs user-defined initialization relating to the low power consumption.
R_LPC_AllModuleClockStop	Stops the clock for all modules.
R_LPC_ChangeSleepModeRetrunClock	Sets the clock source that is selected following release from sleep mode.
R_LPC_Sleep	Transits the low power consumption mode of the MCU to the sleep mode.
R_LPC_DeepSleep	Transits the low power consumption mode of the MCU to the deep sleep mode.
R_LPC_DeepSoftwareStandby	Transits the low power consumption mode of the MCU to the deep software standby mode.
R_LPC_SoftwareStandby	Transits the low power consumption mode of the MCU to the software standby mode.
R_LPC_ChangeOperationPowerControl	Changes the operating power control mode of the MCU.

R_LPC_Create

Performs initialization required to control the low power consumption.

[Syntax]

```
void    R_LPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_Create_UserInit

Performs user-defined initialization relating to the low power consumption.

Remark This API function is called as the R_LPC_Create callback routine.

[Syntax]

```
void    R_LPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_AllModuleClockStop

Stops the clock for all modules.

[Syntax]

```
#include     "r_cg_macrodriver.h"
MD_STATUS    R_LPC_AllModuleClockStop ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeSleepModeRetrunClock

Sets the clock source that is selected following release from sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeSleepModeReturnClock ( return_clock_t clock );
```

[Argument(s)]

I/O	Argument	Description
I	`return_clock_t clock;`	Clock source type RETURN_LOCO: Low-speed on-chip oscillator RETURN_HOCO: High-speed on-chip oscillator RETURN_MAIN_CLOCK: Main clock oscillator RETURN_DISABLE: Switching of the clock source does not proceed.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ARGERROR	Invalid argument *clock* specification

R_LPC_Sleep

Transits the low power consumption mode of the MCU to the sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_Sleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_DeepSleep

Transits the low power consumption mode of the MCU to the deep sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_DeepSleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_DeepSoftwareStandby

Transits the low power consumption mode of the MCU to the deep software standby mode.

[Syntax]

```
MD_STATUS   R_LPC_DeepSoftwareStandby ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_SoftwareStandby

Transits the low power consumption mode of the MCU to the software standby mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_SoftwareStandby ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeOperationPowerControl

Changes the operating power control mode of the MCU.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeOperationPowerControl ( operaring_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	`operating_mode_t mode;`	Operating power control mode type HIGH_SPEED: High-speed operating mode MIDDLE_SPEED: Middle-speed operating mode LOW_SPEED: Low-speed operating mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ERROR2	Change to the middle-speed operating mode is ended abnormally.
MD_ARGERROR	Invalid argument *mode* specification

3.2.6 Interrupt controller (ICU)

Below is a list of API functions output by the Code Generator for interrupt controller use.

Table 3.6 API Functions: [Interrupt Controller]

API Function Name	Function
R_ICU_Create	Performs initialization necessary to control the interrupt controller.
R_ICU_Create_UserInit	Performs user-defined initialization relating to the interrupt controller.
r_icu_irqn_interrupt	Performs processing in response to the external pin interrupts.
r_icu_software_interrupt	Performs processing in response to the software interrupt.
r_icu_software2_interrupt	Performs processing in response to the software interrupt2.
r_icu_nmi_interrupt	Performs processing in response to the NMI pin interrupt.
R_ICU_IRQn_Start	Allows detection of the external pin interrupt.
R_ICU_IRQn_Stop	Prohibits detection of the external pin interrupt.
R_ICU_Software_Start	Allows detection of the software interrupt.
R_ICU_Software2_Start	Allows detection of the software interrupt2.
R_ICU_Software_Stop	Prohibits detection of the software interrupt.
R_ICU_Software2_Stop	Prohibits detection of the software interrupt2.
R_ICU_SoftwareInterrupt_Generate	Generates the software interrupt.
R_ICU_SoftwareInterrupt2_Generate	Generates the software interrupt2.

R_ICU_Create

Performs initialization necessary to control the interrupt controller.

[Syntax]

```
void    R_ICU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt controller.

Remark This API function is called as the R_ICU_Create callback routine.

[Syntax]

```
void    R_ICU_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_irq*n*_interrupt

Performs processing in response to the external pin interrupts.

Remark This API function is called to run interrupt processing for the external pin interrupts.

[Syntax]

```
static  void    r_icu_irqn_interrupt ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

r_icu_software_interrupt

Performs processing in response to the software interrupt.

Remark This API function is called to run the interrupt processing for the software interrupt, which is generated in response to calling of R_ICU_SoftwareInterrupt_Generate.

[Syntax]

```
static  void    r_icu_software_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_software2_interrupt

Performs processing in response to the software interrupt2.

Remark This API function is called to run the interrupt processing for the software interrupt, which is generated in response to calling of R_ICU_SoftwareInterrupt2_Generate.

[Syntax]

```
static  void    r_icu_software2_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_nmi_interrupt

Performs processing in response to the NMI pin interrupt.

Remark This API function is called to run interrupt processing for the NMI pin interrupt.

[Syntax]

```
static  void    r_icu_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Start

Allows detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Start ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Stop

Prohibits detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Stop ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Start

Allows detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software2_Start

Allows detection of the software interrupt 2.

[Syntax]

```
void    R_ICU_Software2_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Stop

Prohibits detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software2_Stop

Prohibits detection of the software interrupt 2.

[Syntax]

```
void    R_ICU_Software2_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_SoftwareInterrupt_Generate

Generates the software interrupt.

Remark r_icu_software_interrupt is called in response to calling od this API function.

[Syntax]

```
void    R_ICU_SoftwareInterrupt_Generate ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_SoftwareInterrupt2_Generate

Generates the software interrupt 2.

Remark r_icu_software2_interrupt is called in response to calling od this API function.

[Syntax]

```
void    R_ICU_SoftwareInterrupt2_Generate ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.7 Buses

Below is a list of API functions output by the Code Generator for buses use.

Table 3.7 API Functions: [Buses]

API Function Name	Function
R_BSC_Create	Performs initialization necessary to control the buses.
R_BSC_Create_UserInit	Performs user-defined initialization relating to the buses.
r_bsc_buserr_interrupt	Performs processing in response to the bus error (illegal address access).
R_BSC_Error_Monitoring_Start	Allows the detection of bus errors (illegal address access).
R_BSC_Error_Monitoring_Stop	Prohibits the detection of bus errors (illegal address access).
R_BSC_InitializeSDRAM	Performs initialization of SDRAM controller.

R_BSC_Create

Performs initialization necessary to control the buses.

[Syntax]

```
void    R_BSC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Create_UserInit

Performs user-defined initialization relating to the buses.

Remark This API function is called as the R_BSC_Create callback routine.

[Syntax]

```
void    R_BSC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bsc_buserr_interrupt

Performs processing in response to the bus error (illegal address access).

Remarks 1. This API function is called to run interrupt processing for a bus error (illegal address access), which is generated through access by the processing program to a location within an illegal address range.

Remarks 2. The bus master that caused the bus error can be confirmed by reading the MST bit of bus error status register 1 (BERSR1) from within this API function.

Remarks 3. The illegal address (high-order 13 bits) that caused the bus error can be confirmed by reading the ADDR bit of bus error status register 2 (BERSR2) from within this API function.

[Syntax]

```
static  void    r_bsc_buserr_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Start

Allows the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Stop

Prohibits the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_InitializeSDRAM

Performs initialization of SDRAM controller.

[Syntax]

```
void    R_BSC_InitializeSDRAM ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.8 DMA Controller(DMAC)

Below is a list of API functions output by the Code Generator for DMA controller use.

Table 3.8 API Functions: [DMA Controller]

API Function Name	Function
R_DMAC_Create	Performs initialization necessary to control the DMA controller.
R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
r_dmac_dmacni_interrupt	Performs processing in response to the transfer end interrupt.
r_dmacn_callback_transfer_end	Performs processing in response to the transfer end interrupt.
r_dmacn_callback_transfer_escape_end	Performs processing in response to the escape transfer end interrupt.
R_DMACn_Start	Allows starting of the DMAC controller.
R_DMACn_Stop	Prohibits starting of the DMAC controller.
R_DMACn_Set_SoftwareTrigger	Sets the software request of DMA transfer by software.
R_DMACn_Clear_SoftwareTrigger	Clears the software request of DMA transfer by software.

R_DMAC_Create

Performs initialization necessary to control the DMA controller.

[Syntax]

```
void    R_DMAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the R_DMAC_Create callback routine.

[Syntax]

```
void    R_DMAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmac_dmac*n*i_interrupt

Performs processing in response to the transfer end interrupt.

[Syntax]

```
static  void    r_dmac_dmacni_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dmac*n*_callback_transfer_end

Performs processing in response to the transfer end interrupt.

Remark This API function is called as thecall r_dmac_dmacni_interrupt back routine.

[Syntax]

```
static  void    r_dmacn_callback_transfer_end ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dmac*n*_callback_transfer_escape_end

Performs processing in response to the escape transfer end interrupt.

Remark This API function is called as the r_dmac_dmacni_interrupt callback routine, which is generated by escape transfer end.

[Syntax]

```
static  void    r_dmacn_callback_transfer_escape_end ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Start

Allows starting of the DMAC controller.

[Syntax]

```
void    R_DMACn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Stop

Prohibits starting of the DMAC controller.

[Syntax]

```
void    R_DMACn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Set_SoftwareTrigger

Sets the software request of DMA transfer by software.

[Syntax]

```
void    R_DMACn_Set_SoftwareTrigger ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Clear_SoftwareTrigger

Clears the software request of DMA transfer by software.

[Syntax]

```
void    R_DMACn_Clear_SoftwareTrigger ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.9 Data transfer controller (DTC)

Below is a list of API functions output by the Code Generator for data transfer controller use.

Table 3.9 API Functions: [Data Transfer Controller]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the data transfer controller.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
R_DTCm_Start	Allows starting of the data transfer controller.
R_DTCm_Stop	Prohibits starting of the data transfer controller.

R_DTC_Create

Performs initialization necessary to control the data transfer controller.

[Syntax]

```
void    R_DTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC_Create_UserInit

Performs user-defined initialization relating to the data transfer controller.

Remark This API function is called as the R_DTC_Create callback routine.

[Syntax]

```
void    R_DTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Start

Allows starting the data transfer controller.

Remark In this API function, starting the data transfer controller is allowed by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.
m is the transfer data number, *n* is the interrupt vector number.

[Syntax]

```
void    R_DTCm_Start ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Stop

Prohibits starting of the data transfer controller.

Remark In this API function, starting the data transfer controller is prohibited by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.
m is the transfer data number, *n* is the interrupt vector number.

[Syntax]

```
void    R_DTCm_Stop ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

3.2.10 Event link controller (ELC)

Below is a list of API functions output by the Code Generator for event link controller use.

Table 3.10 API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
r_elc_elsrni_interrupt	Performs processing in response to the event link interrupt.
R_ELC_Start	Starts interlinked operation of peripheral functions.
R_ELC_Stop	Ends interlinked operation of peripheral functions.
R_ELC_GenerateSoftwareEvent	Generates the software event.
R_ELC_Set_PortBuffern	Sets the value of a port buffer.
R_ELC_Get_PortBuffern	Gets the value of a port buffer.

R_ELC_Create

Performs initialization necessary to control the event link controller.

[Syntax]

```
void    R_ELC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Create_UserInit

Performs user-defined initialization relating to the event link controller.

Remark This API function is called as the R_ELC_Create callback routine.

[Syntax]

```
void    R_ELC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_elc_elsr*n*i_interrupt

Performs processing in response to the event link interrupt.

Remark This API function is called to run interrupt processing for the event signal defined in event link setting register.
n is the source number.

[Syntax]

```
static  void    r_elc_elsrni_interrupt ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ELC_Start

Starts interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Ends interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_GenerateSoftwareEvent

Generates the software event.

[Syntax]

```
void    R_ELC_GenerateSoftwareEvent ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Set_PortBuffer*n*

Sets the value of a port buffer.

[Syntax]

```
void    R_ELC_Set_PortBuffern ( uint8_t value );
```

Remark *n* is the source number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t value;`	The value set in the port buffer.

[Return value]

None.

R_ELC_Get_PortBuffer*n*

Gets the value of a port buffer.

[Syntax]

```
void    R_ELC_Get_PortBuffern ( uint8_t * const value );
```

Remark *n* is the source number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const value;`	Pointer to the location where the obtained value is to be stored.

[Return value]

None.

3.2.11 I/O ports

Below is a list of API functions output by the Code Generator for I/O ports use.

Table 3.11 API Functions: [I/O Ports]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control the I/O ports.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the I/O ports.

R_PORT_Create

Performs initialization necessary to control the I/O ports.

[Syntax]

```
void    R_PORT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the I/O ports.

Remark This API function is called as the R_PORT_Create callback routine.

[Syntax]

```
void    R_PORT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.12 Multi-function timer pulse unit 2 (MTU2)

Below is a list of API functions output by the Code Generator for multi-function timer pulse unit 2 use.

Table 3.12 API Functions: [Multi-Function Timer Pulse Unit 2]

API Function Name	Function
R_MTU2_Create	Performs initialization necessary to control the multi-function timer pulse unit 2.
R_MTU2_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit 2.
r_mtu2_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu2_cj_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu2_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu2_cj_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu2_tciun_interrupt	Performs processing in response to the underflow interrupt.
R_MTU2_Cn_Start	Starts counting by a 16-bit timer.
R_MTU2_Cn_Stop	Ends counting by a 16-bit timer.

R_MTU2_Create

Performs initialization necessary to control the multi-function timer pulse unit 2.

[Syntax]

```
void    R_MTU2_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU2_Create_UserInit

Performs user-defined initialization relating to the multi-function timer pulse unit 2.

Remark This API function is called as the R_MTU2_Create callback routine.

[Syntax]

```
void    R_MTU2_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 2 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static  void    r_mtu2_tgimn_interrupt ( void );
```

Remark *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_c*j*_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 2 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static  void    r_mtu2_cj_tgimn_interrupt ( void );
```

Remark *j* is the relationship channel number, *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu2_tcivn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_c*j*_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu2_cj_tcivn_interrupt ( void );
```

Remark *j* is the relationship channel number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tciu*n*_interrupt

Performs processing in response to the underflow interrupt.

Remark This API function is called to run interrupt processing for the underflow interrupt, which is generated in response to an underflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu2_tciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_C*n*_Start

Starts counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_Cn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_C*n*_Stop

Ends counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_Cn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.13 Multi-function timer pulse unit 3 (MTU3)

Below is a list of API functions output by the Code Generator for multi-function timer pulse unit 3 use.

Table 3.13 API Functions: [Multi-Function Timer Pulse Unit 3]

API Function Name	Function
R_MTU3_Create	Performs initialization necessary to control the multi-function timer pulse unit 3.
R_MTU3_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit 3.
r_mtu3_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu3_cj_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu3_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu3_cj_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu3_tciun_interrupt	Performs processing in response to the underflow interrupt.
R_MTU3_Cn_Start	Starts counting by a 16-bit timer.
R_MTU3_Cn_Stop	Ends counting by a 16-bit timer.

R_MTU3_Create

Performs initialization necessary to control the multi-function timer pulse unit 3.

[Syntax]

```
void    R_MTU3_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU3_Create_UserInit

Performs user-defined initialization relating to the multi-function timer pulse unit 3.

Remark This API function is called as the R_MTU3_Create callback routine.

[Syntax]

```
void    R_MTU3_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 3 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static  void    r_mtu3_tgimn_interrupt ( void );
```

Remark *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c*j*_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 3 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static  void    r_mtu3_cj_tgimn_interrupt ( void );
```

Remark *j* is the relationship channel number, *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu3_tcivn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c*j*_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu3_cj_tcivn_interrupt ( void );
```

Remark *j* is the relationship channel number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciu*n*_interrupt

Performs processing in response to the underflow interrupt.

Remark This API function is called to run interrupt processing for the underflow interrupt, which is generated in response to an underflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu3_tciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*n*_Start

Starts counting by the 16-bit timer.

[Syntax]

```
void    R_MTU3_Cn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*n*_Stop

Ends counting by the 16-bit timer.

[Syntax]

```
void    R_MTU3_Cn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.14 Port output enable 2 (POE2)

Below is a list of API functions output by the Code Generator for port output enable 2 use.

Table 3.14 API Functions: [Port Output Enable 2]

API Function Name	Function
R_POE2_Create	Performs initialization necessary to control the port output enable 2.
R_POE2_Create_UserInit	Performs user-defined initialization relating to the port output enable 2.
r_poe2_oein_interrupt	Performs processing in response to the output enable interrupt n (OEI*n*).
R_POE2_Start	Places the MTU's complementary PWM output pins in the high-impedance state.
R_POE2_Stop	Releases the R_POE2_Stop MTU's complementary PWM output pins from the high-impedance state.
R_POE2_Set_HiZ_MTUn	Sets the high-impedance state for the MTUn pins.
R_POE2_Clear_HiZ_MTUn	Clear the high-impedance state for the MTUn pins

R_POE2_Create

Performs initialization necessary to control the port output enable 2.

[Syntax]

```
void    R_POE2_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Create_UserInit

Performs user-defined initialization relating to the port output enable 2.

Remark This API function is called as the R_POE2_Create callback routine.

[Syntax]

```
void    R_POE2_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_poe2_oei*n*_interrupt

Performs processing in response to the output enable interrupt *n* (OEI*n*).

Remark This API function is called to run interrupt processing for the output enable interrupt *n* (OEI*n*), which is generated when a pin (any of POE0#, POE1#, POE2#, POE3#, and POE8#) becomes high-impedance or the output short flag 1 is set.

[Syntax]

```
static  void    r_poe2_oein_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE2_Start

Places the MTU's complementary PWM output pins in the high-impedance state.

[Syntax]

```
void    R_POE2_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Stop

Releases the MTU's complementary PWM output pins from the high-impedance state.

[Syntax]

```
void    R_POE2_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Set_HiZ_MTU*n*

Sets the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE2_Set_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE2_Clear_HiZ_MTU*n*

Clear the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE2_Clear_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.15 Port output enable 3 (POE3)

Below is a list of API functions output by the Code Generator for port output enable 3 use.

Table 3.15 API Functions: [Port Output Enable 3]

API Function Name	Function
R_POE3_Create	Performs initialization necessary to control the port output enable 3.
R_POE3_Create_UserInit	Performs user-defined initialization relating to the port output enable 3.
r_poe3_oein_interrupt	Performs processing in response to the output enable interrupt n (OEI*n*).
R_POE3_Start	Places the MTU's complementary PWM output pins in the high-impedance state.
R_POE3_Stop	Releases the R_POE3_Stop MTU's complementary PWM output pins from the high-impedance state.
R_POE3_Set_HiZ_MTUn	Sets the high-impedance state for the MTU*n* pins.
R_POE3_Clear_HiZ_MTUn	Clear the high-impedance state for the MTU*n* pins.
R_POE3_Set_HiZ_GPTn	Sets the high-impedance state for the GPT*n* pins.
R_POE3_Clear_HiZ_GPTn	Clear the high-impedance state for the GPT*n* pins.

R_POE3_Create

Performs initialization necessary to control the port output enable 3.

[Syntax]

```
void    R_POE3_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Create_UserInit

Performs user-defined initialization relating to the port output enable 3.

Remark This API function is called as the R_POE3_Create callback routine.

[Syntax]

```
void    R_POE3_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_poe3_oei*n*_interrupt

Performs processing in response to the output enable interrupt.

Remark This API function is called to run interrupt processing for the output enable interrupt, which is generated when a related pin becomes high-impedance or the output short flag 1 is set.

[Syntax]

```
static  void    r_poe3_oein_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Start

Places the related pins in the high-impedance state.

[Syntax]

```
void    R_POE3_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Stop

Replaces the related output pins from the high-impedance state.

[Syntax]

```
void    R_POE3_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_MTU*n*

Sets the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE3_Set_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_MTU*n*

Clear the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE3_Clear_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_GPT*n*

Sets the high-impedance state for the GPT*n* pins.

[Syntax]

```
void    R_POE3_Set_HiZ_GPTn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_GPT*n*

Clear the high-impedance state for the GPT*n* pins.

[Syntax]

```
void    R_POE3_Clear_HiZ_GPTn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.16 General PWM timer (GPT)

Below is a list of API functions output by the Code Generator for general PWM timer use.

Table 3.16 API Functions: [General PWM timer]

API Function Name	Function
R_GPT_Create	Performs initialization necessary to control the general PWM timer.
R_GPT_Create_UserInit	Performs user-defined initialization relating to the general PWM timer.
r_gpt_gtcimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_gpt_gtcivn_interrupt	Performs processing in response to the overflow interrupt.
r_gpt_gtciun_interrupt	Performs processing in response to the underflow interrupt.
r_gpt_gdten_interrupt	Performs processing in response to the dead time error interrupt.
r_gpt_etgip_interrupt	Performs processing in response to the external trigger rising interrupt.
r_gpt_etgin_interrupt	Performs processing in response to the external trigger falling interrupt.
R_GPTn_Start	Starts counting by a 16-bit timer.
R_GPTn_Stop	Ends counting by a 16-bit timer.
R_GPTn_HardwareStart	Allows GPT interrupts.
R_GPTn_HardwareStop	Prohibits GPT interrupts.

R_GPT_Create

Performs initialization necessary to control the general PWM timer.

[Syntax]

```
void    R_GPT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_GPT_Create_UserInit

Performs user-defined initialization relating to the general PWM timer.

Remark This API function is called as the R_GPT_Create callback routine.

[Syntax]

```
void    R_GPT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtci*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

[Syntax]

```
static  void    r_gpt_gtcimn_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciv*n*_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static  void    r_gpt_gtcivn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciu*n*_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
static  void    r_gpt_gtciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gdte*n*_interrupt

Performs processing in response to the dead time error interrupt.

[Syntax]

```
static  void    r_gpt_gdten_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgip_interrupt

Performs processing in response to the external trigger rising interrupt.

[Syntax]

```
static  void    r_gpt_eigip_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgin_interrupt

Performs processing in response to the external trigger falling interrupt.

[Syntax]

```
static  void    r_gpt_etgin_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_GPTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_GPTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStart

Allows detection of GPT interrupts.

Remark This API function enables GPT interrupts when starting timer count by the hardware trigger.

[Syntax]

```
void    R_GPTn_HardwareStart ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStop

Prohibits detection of GPT interrupts.

Remark This API function disables GPT interrupts when starting timer count by the hardware trigger.

[Syntax]

```
void    R_GPTn_HardwareStop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.17 16-bit timer pulse unit (TPU)

Below is a list of API functions output by the Code Generator for 16-bit timer pulse unit use.

Table 3.17 API Functions: [16-bit timer pulse unit]

API Function Name	Function
R_TPU_Create	Performs initialization necessary to control the 16-bit timer pulse unit.
R_TPU_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer pulse unit.
r_tpu_tginm_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_tpu_tcinv_interrupt	Performs processing in response to the overflow interrupt.
r_tpu_tcinu_interrupt	Performs processing in response to the underflow interrupt.
R_TPUn_Start	Starts counting by a 16-bit timer.
R_TPUn_Stop	Ends counting by a 16-bit timer.

R_TPU_Create

Performs initialization necessary to control the 16-bit timer pulse unit.

[Syntax]

```
void    R_TPU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TPU_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer pulse unit.

Remark This API function is called as the R_TPU_Create callback routine.

[Syntax]

```
void    R_TPU_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*nm*_interrupt

Performs processing in response to the input capture/compare match interrupt.

[Syntax]

```
static  void    r_tpu_tginm_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*v_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static  void    r_tpu_tcinv_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*u_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
static  void    r_tpu_tcinu_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_TPUn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_TPUn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.18 8-bit timer (TMR)

Below is a list of API functions output by the Code Generator for 8-bit timer use.

Table 3.18 API Functions: [8-bit timer]

API Function Name	Function
R_TMR_Create	Performs initialization necessary to control the 8-bit timer pulse unit.
R_TMR_Create_UserInit	Performs user-defined initialization relating to the 8-bit timer pulse unit.
r_tmr_cmimn_interrupt	Performs processing in response to the compare match interrupt.
r_tmr_ovin_interrupt	Performs processing in response to the overflow interrupt.
R_TMRn_Start	Starts counting by an 8-bit timer.
R_TMRn_Stop	Ends counting by an 8-bit timer.

R_TMR_Create

Performs initialization necessary to control the 8-bit timer.

[Syntax]

```
void    R_TMR_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_Create_UserInit

Performs user-defined initialization relating to the 8-bit timer.

Remark This API function is called as the R_TMR_Create callback routine.

[Syntax]

```
void    R_TMR_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_cmi*mn*_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
static  void    r_tmr_cmimn_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_ovi*n*_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static  void    r_tmr_ovin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR*n*_Start

Starts counting by an 8-bit timer.

[Syntax]

```
void    R_TMRn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR*n*_Stop

Ends counting by an 8-bit timer.

[Syntax]

```
void    R_TMRn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.19 Programmable pulse generator (PPG)

Below is a list of API functions output by the Code Generator for programmable pulse generator use.

Table 3.19 API Functions: [Programmable Pulse Generator]

API Function Name	Function
R_PPG_Create	Performs initialization necessary to control the programmable pulse generator .
R_PPG_Create_UserInit	Performs user-defined initialization relating to the programmable pulse generator .

R_PPG_Create

Performs initialization necessary to control the programmable pulse generator .

[Syntax]

```
void    R_PPG_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PPG_Create_UserInit

Performs user-defined initialization relating to the programmable pulse generator .

Remark This API function is called as the R_PPG_Create callback routine.

[Syntax]

```
void    R_PPG_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.20 Compare match timer (CMT)

Below is a list of API functions output by the Code Generator for compare match timer use.

Table 3.20 API Functions: [Compare Match Timer]

API Function Name	Function
R_CMTn_Create	Performs initialization necessary to control the compare match timer.
R_CMTn_Create_UserInit	Performs user-defined initialization relating to the compare match timer.
r_cmt_cmin_interrupt	Performs processing in response to the compare match interrupt (CMI*n*).
R_CMTn_Start	Starts counting by a 16-bit timer.
R_CMTn_Stop	Ends counting by a 16-bit timer.

R_CMT*n*_Create

Performs initialization necessary to control the compare match timer.

[Syntax]

```
void    R_CMTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Create_UserInit

Performs user-defined initialization relating to the compare match timer.

Remark This API function is called as the R_CMTn_Create callback routine.

[Syntax]

```
void    R_CMTn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmt_cmi*n*_interrupt

Performs processing in response to the compare match interrupt (CMI*n*).

Remark This API function is called to run interrupt processing for the compare match interrupt (CMI*n*), which is generated because the current counter value (value of the compare match timer counter, CMCR) matched the defined counter value (value of the compare match timer constant register, CMCOR).

[Syntax]

```
static  void    r_cmt_cmin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.21 Compare match timer W (CMTW)

Below is a list of API functions output by the Code Generator for compare match timer W use.

Table 3.21 API Functions: [Compare Match Timer W]

API Function Name	Function
R_CMTWn_Create	Performs initialization necessary to control the compare match timer W.
R_CMTWn_Create_UserInit	Performs user-defined initialization relating to the compare match timer W.
r_cmtw_cmwin_interrupt	Performs processing in response to the compare match interrupt.
r_cmtw_icmin_interrupt	Performs processing in response to the input capture interrupt.
r_cmtw_ocmin_interrupt	Performs processing in response to the output compare interrupt.
R_CMTWn_Start	Starts counting by a 16-bit timer.
R_CMTWn_Stop	Ends counting by a 16-bit timer.

R_CMTW*n*_Create

Performs initialization necessary to control the compare match timer W.

[Syntax]

```
void    R_CMTPn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Create_UserInit

Performs user-defined initialization relating to the compare match timer W.

Remark This API function is called as the R_CMTWn_Create callback routine.

[Syntax]

```
void    R_CMTWn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_cmwi*n*_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
static  void    r_cmtw_cmwin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_ic*min*_interrupt

Performs processing in response to the input capture interrupt.

[Syntax]

```
static  void    r_cmtw_icmin_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_oc*min*_interrupt

Performs processing in response to the output compare interrupt.

[Syntax]

```
static  void    r_cmtw_ocmin_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_CMTWn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_CMTWn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.22 Realtime clock (RTC)

Below is a list of API functions output by the Code Generator for realtime clock use.

Table 3.22 API Functions: [Realtime Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control the realtime clock.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the realtime clock.
r_rtc_alm_interrupt	Performs processing in response to the alarm interrupt (ALM).
r_rtc_prd_interrupt	Performs processing in response to the periodic interrupt (PRD).
r_rtc_cup_interrupt	Performs processing in response to the carry interrupt (CUP).
R_RTC_Set_CalendarAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).
R_RTC_Set_BinaryAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).
R_RTC_Set_ConstPeriodInterruptOn	Sets the period of the periodic interrupt (PRD) and allows detection of PRD.
R_RTC_Set_ConstPeriodInterruptOff	Prohibits detection of the periodic interrupt (PRD).
R_RTC_Set_CarryInterruptOn	Allows detection of the carry interrupt (CUP).
R_RTC_Set_CarryInterruptOff	Prohibits detection of the carry interrupt (CUP).
R_RTC_Set_RTCOUTOn	Set the RTCOUT output period and starts RTCOUT output.
R_RTC_Set_RTCOUTOff	Ends the RTCOUT output.
R_RTC_Start	Starts counting.
R_RTC_Stop	Ends counting.
R_RTC_Restart	Initializes the counter then starts counting.
R_RTC_Set_CalendarCounterValue	Sets the values of the calendar and time counters.
R_RTC_Get_CalendarCounterValue	Gets the values of the calendar and time counters.
R_RTC_Set_BinaryCounterValue	Sets the value of the binary counter.
R_RTC_Get_BinaryCounterValue	Gets the value of the binary counter.
R_RTC_Get_CalendarTimeCaptureValuen	Gets the captured calendar time value.
R_RTC_Get_BinaryTimeCaptureValuen	Gets the captured binary time value.

R_RTC_Create

Performs initialization necessary to control the realtime clock.

[Syntax]

```
void    R_RTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the realtime clock.

Remark This API function is called as the R_RTC_Create callback routine.

[Syntax]

```
void    R_RTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_alm_interrupt

Performs processing in response to the alarm interrupt (ALM).

Remark This API function is called to run interrupt processing for the alarm interrupt (ALM), which is generated when the condition specified by R_RTC_Set_CalendarAlarm is satisfied.

[Syntax]

```
static  void    r_rtc_alm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_prd_interrupt

Performs processing in response to the periodic interrupt (PRD).

Remark This API function is called to run interrupt processing for the periodic interrupt (PRD), which is generated when the period specified by R_RTC_Set_ConstPeriodInterruptOn elapses.

[Syntax]

```
static  void    r_rtc_prd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_cup_interrupt

Performs processing in response to the carry interrupt (CUP).

Remark This API function is called to run interrupt processing for the carry interrupt (CUP), which is generated on carries from the seconds counter (RSECCNT) or binary counter 0 (BCNT0) or when the 64-Hz counter (R64CNT) is read at the same time as a carry from the 64-Hz counter (R64CNT).

[Syntax]

```
static  void    r_rtc_cup_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CalendarAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarAlarm ( rtc_calendar_alarm_enable_t alarm_enable,
rtc_calendar_alarm_value_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_calendar_alarm_enable_t` *`alarm_enable`*`;`	Comparison flags (year, month, date, day-of-the-week, hour, minute, and second). 0x0: Comparison proceeds 0x80: Comparison does not proceed
I	`rtc_calendar_alarm_value_t` *`alarm_val`*`;`	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remarks 1. The configuration of the comparison flag structure rtc_calendar_alarm_enable_t is shown below.

```
typedef struct {
    uint8_t sec_enb;    /* Second */
    uint8_t min_enb;    /* Minute */
    uint8_t hr_enb;     /* Time */
    uint8_t day_enb;    /* Date */
    uint8_t wk_enb;     /* Day-of-week */
    uint8_t mon_enb;    /* Month */
    uint8_t yr_enb;     /* Year */
} rtc_calendar_alarm_enable_t;
```

Remarks 2. The configuration of the calendar and time values rtc_calendar_alarm_value_t is shown below.

```
typedef struct {
    uint8_t rsecar;     /* second */
    uint8_t rminar;     /* Minute */
    uint8_t rhrar;      /* Time */
    uint8_t rdayar;     /* Date */
    uint8_t rwkar;      /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmonar;     /* Month */
    uint16_t ryrar;     /* Year */
} rtc_calendar_alarm_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Set_BinaryAlarm ( uint32_t alarm_enable, uint32_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *alarm_enable*;	Comparison flag 0x0: Comparison does not proceed 0x1: Comparison proceeds
I	uint32_t *alarm_val*;	The value of the binary counter

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOn

Sets the period of the periodic interrupt (PRD) and allows detection of the PRD.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_int_period_t period;`	Period of the periodic interrupt (PRD). PES_2_SEC: 2 seconds PES_1_SEC: 1 second PES_1_2_SEC: 1/2 second PES_1_4_SEC: 1/4 second PES_1_8_SEC: 1/8 second PES_1_16_SEC: 1/16 second PES_1_32_SEC: 1/32 second PES_1_64_SEC: 1/64 second PES_1_128_SEC: 1/128 second PES_1_256_SEC: 1/256 second

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOff

Prohibits detection of the periodic interrupt (PRD).

[Syntax]

```
void    R_RTC_Set_ConstPeriodInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOn

Allows detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOff

Prohibits detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOn

Sets the RTCOUT output period and starts RTCOUT output.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_RTCOUTOn ( rtc_rtcout_period_t rtcout_freq );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_rtcout_period_t` *`rtcout_freq`*`;`	RTCOUT output period RTCOUT_1HZ: 1Hz RTCOUT_64HZ: 64Hz

[Return value]

None.

R_RTC_Set_RTCOUTOff

Ends RTCOUT output.

[Syntax]

```
void    R_RTC_Set_RTCOUTOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts counting.

[Syntax]

```
void    R_RTC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends counting.

[Syntax]

```
void    R_RTC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Restart

Initializes the counter then starts counting.

Remarks 1. When the realtime clock is operating in the calendar counting mode, this API function initializes the counters to the values specified by the argument *counter_write_val*.

Remarks 2. When the realtime clock is operating in the binary counting mode, this API function ignores the value specified by the argument *counter_write_val* and clears the counter to zero.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Restart ( rtc_calendarcounter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_calendarcounter_value_t counter_write_val;`	Initial value (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the initial value rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_CalendarCounterValue

Sets the calendar and time values.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarCounterValue ( rtc_calendarcounter_value_t counter_write_val
);
```

[Argument(s)]

I/O	Argument	Description
I	rtc_calendarcounter_value_t *counter_write_val*;	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Get_CalendarCounterValue

Gets the calendar and time values.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Get_CalendarCounterValue ( rtc_calendarcounter_value_t * const
counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	`rtc_calendarcounter_value_t * const counter_read_val;`	Pointer to the area where the obtained calendar and time values (year, month, date, day-of-week, time, minute, and second) are to be stored

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryCounterValue

Sets the value of the binary counter.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Set_BinaryCounterValue ( uint32_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t counter_write_val;`	The value of the binary counter

[Return value]

None.

R_RTC_Get_BinaryCounterValue

Gets the value of the binary count.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Get_BinaryCounterValue ( uint32_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const` *`counter_read_val`*`;`	Pointer to an area where the obtained value of the binary counter is to be stored

[Return value]

None.

R_RTC_Get_CalendarTimeCaptureValue*n*

Gets the captured calendar time value.

[Syntax]

```
void    R_RTC_Get_CalendarTimeCaptureValuen ( rtc_calendarcounter_value_t  * const
counter_read_val  );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	rtc_calendarcounter_value_t * const *counter_read_val;*	Pointer to the area where the obtained calendar and time values.

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;        /* Second */
    uint8_t rmincnt;        /* Minute */
    uint8_t rhrcnt;         /* Time */
    uint8_t rdaycnt;        /* Date */
    uint8_t rwkcnt;         /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;        /* Month */
    uint16_t ryrcnt;        /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Get_BinaryTimeCaptureValue*n*

Gets the value of the binary count.

[Syntax]

```
void    R_RTC_Get_BinaryTimeCaptureValuen ( uint32_t  * const  counter_read_val   );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const counter_read_val;`	The value of the binary counter.

[Return value]

None.

3.2.23 Watchdog timer (WDT)

Below is a list of API functions output by the Code Generator for watchdog timer use.

Table 3.23 API Functions: [Watchdog Timer]

API Function Name	Function
R_WDT_Create	Performs initialization necessary to control the watchdog timer.
R_WDT_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
r_wdt_nmi_interrupt	Performs processing in response to the non-maskable interrupt.
r_wdt_wuni_interrupt	Performs processing in response to the non-maskable/maskable interrupt.
R_WDT_Restart	Clears the watchdog timer counter and resumes counting.

R_WDT_Create

Performs initialization necessary to control the watchdog timer.

[Syntax]

```
void    R_WDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Create_UserInit

Performs user-defined initialization relating to the watchdog timer.

Remark This API function is called as the R_WDT_Create callback routine.

[Syntax]

```
void    R_WDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt , which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static  void    r_wdt_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wdt_wuni_interrupt

Performs processing in response to the non-maskable/maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable / maskable interrupt , which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static  void    r_wdt_wuni_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```
void    R_WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.24 Independent watchdog timer (IWDT)

Below is a list of API functions output by the Code Generator for independent watchdog timer use.

Table 3.24 API Functions: [Independent Watchdog Timer]

API Function Name	Function
R_IWDT_Create	Performs initialization necessary to control the independent watchdog timer.
R_IWDT_Create_UserInit	Performs user-defined initialization relating to the independent watchdog timer.
r_iwdt_nmi_interrupt	Performs processing in response to the non-maskable interrupt (WUNI).
r_iwdt_iwuni_interrupt	Performs processing in response to the non-maskable/maskable interrupt.
R_IWDT_Restart	Clears the independent watchdog timer counter and resumes counting.

R_IWDT_Create

Performs initialization necessary to control the independent watchdog timer.

[Syntax]

```
void    R_IWDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Create_UserInit

Performs user-defined initialization relating to the independent watchdog timer.

Remark This API function is called as the R_IWDT_Create callback routine.

[Syntax]

```
void    R_IWDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_iwdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt (WUNI).

Remark This API function is called to run interrupt processing for the non-maskable interrupt (WUNI), which is generated when the down-counter underflows or refreshing proceeds outside the period where it is permitted.

[Syntax]

```
static  void    r_iwdt_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_iwdt_iwuni_interrupt

Performs processing in response to the non-maskable/maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable / maskable interrupt , which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static  void    r_iwdt_iwuni_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Restart

Clears the independent watchdog timer counter and resumes counting.

[Syntax]

```
void    R_IWDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.25 Serial communications interface (SCI)

Below is a list of API functions output by the Code Generator for serial communications interface use.

Table 3.25 API Functions: [Serial Communications Interface]

API Function Name	Function
R_SCIn_Create	Performs initialization necessary to control the serial communications interface.
R_SCIn_Create_UserInit	Performs user-defined initialization related to the serial communications interface.
r_scin_transmitend_interrupt	Performs processing in response to the transmit-end interrupts.
r_scin_transmit_interrupt	Performs processing in response to the transmit-data-empty interrupts.
r_scin_receive_interrupt	Performs processing in response to the receive-data-full interrupts.
r_scin_receiveerror_interrupt	Performs processing in response to the receive error interrupts.
R_SCIn_Start	Starts SCI communication.
R_SCIn_Stop	Ends SCI communication.
R_SCIn_Serial_Send	Starts SCI transmission (synchronous mode).
R_SCIn_Serial_Receive	Starts SCI reception (synchronous mode).
R_SCIn_Serial_Multiprocessor_Send	Starts SCI transmission (multi-processor communications function).
R_SCIn_Serial_Multiprocessor_Receive	Starts SCI reception (multi-processor communications function).
R_SCIn_Serial_Send_Receive	Starts SCI transmission/reception (clock synchronous mode).
R_SCIn_SmartCard_Send	Starts SCI transmission (smart card interface mode).
R_SCIn_SmartCard_Receive	Starts SCI reception (smart card interface mode).
R_SCIn_IIC_Master_Send	Starts SCI master transmission (simple I^2C mode).
R_SCIn_IIC_Master_Receive	Starts SCI master reception (simple I^2C mode).
R_SCIn_SPI_Master_Send	Starts SCI master transmission (simple SPI mode).
R_SCIn_SPI_Master_Send_Receive	Starts SCI master transmission/reception (simple SPI mode).
R_SCIn_SPI_Slave_Send	Starts SCI slave transmission (simple SPI mode).
R_SCIn_SPI_Slave_Send_Receive	Starts SCI slave transmission/reception (simple SPI mode).
R_SCIn_IIC_StartCondition	Sends the start bit.
R_SCIn_IIC_StopCondition	Sends the stop bit.
r_scin_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scin_callback_receiveend	Performs processing in response to the receive-data-full interrupts.
r_scin_callback_receiveerror	Performs processing in response to the receive error interrupts.

R_SCI*n*_Create

Performs initialization necessary to control the serial communication interface.

[Syntax]

```
void    R_SCIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Create_UserInit

Performs user-defined initialization related to the serial communications interface.

Remark This API function is called as the R_SCIn_Create callback routine.

[Syntax]

```
void    R_SCIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmitend_interrupt

Performs processing in response to the transmit-end interrupts.

Remark This API function is called to run interrupt processing for the transmit-end interrupts.

[Syntax]

```
static  void    r_scin_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmit_interrupt

Performs processing in response to the transmit-data-empty interrupts.

Remark This API function is called to run interrupt processing for the transmit-data-empty interrupts.

[Syntax]

```
static  void    r_scin_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receive_interrupt

Performs processing in response to the receive-data-full interrupts.

Remark This function is called to run interrupt processing for the receive-data-full interrupts.

[Syntax]

```
static  void    r_scin_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receiveerror_interrupt

Performs processing in response to the receive error interrupts.

Remark This API function is called to run interrupt processing for the receive error interrupts.

[Syntax]

```
static  void    r_scin_receiveerror_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Start

Starts SCI communication.

[Syntax]

```
void    R_SCIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Stop

Ends SCI communication.

[Syntax]

```
void    R_SCIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Serial_Send

Starts SCI transmission (asynchronous mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Receive

Starts SCI reception (asynchronous mode).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remarks 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCIn_Serial_Multiprocessor_Send

Starts SCI transmission (multi-processor communications function).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Multiprocessor_Send ( uint8_t * id_buf, uint16_t id_num,
uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *        id_buf;`	Pointer to a buffer storing the transmission ID
I	`uint16_t         id_num;`	Total amount of ID to send
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t         tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Multiprocessor_Receive

Starts SCI reception (multi-processor communications function).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remarks 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Multiprocessor_Receive ( uint8_t * const rx_buf, uint16_t
rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_Serial_Send_Receive

Starts SCI transmission/reception (clock synchronous mode).

Remarks 1. This API function repeats SCI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function repeats SCI reception processing in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remarks 3. When performing a SCI transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Send

Starts SCI transmission (smart card interface mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include     "r_cg_macrodriver.h"
MD_STATUS    R_SCIn_SmartCard_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Receive

Starts SCI reception (smart card interface mode).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remarks 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SmartCard_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_IIC_Master_Send

Starts SCI master transmission (simple I^2C mode).

Remarks 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. SCI master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master transmission.

Remarks 3. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_SCIn_IIC_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num
);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t          adr;`	Slave address
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t         tx_num;`	Total amount of data to send

[Return value]

None.

R_SCI*n*_IIC_Master_Receive

Starts SCI master reception (simple I2C mode).

Remarks 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr*. SCI master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remarks 2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master reception.

Remarks 3. When performing a SCI master reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_SCIn_IIC_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t 
rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

None.

R_SCI*n*_SPI_Master_Send

Starts SCI master transmission (simple SPI mode).

Remarks 1. This API function repeats the byte-level SCI master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Master_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Master_Send_Receive

Starts SCI master transmission/reception (simple SPI mode).

Remarks 1. This API function repeats SCI master transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function repeats SCI master reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remarks 3. When performing a SCI master transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Master_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t         tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t         rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send

Starts SCI slave transmission (simple SPI mode).

Remarks 1. This API function repeats the byte-level SCI slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI slave transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send_Receive

Starts SCI slave transmission/reception (simple SPI mode).

Remarks 1. This API function repeats SCI slave transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function repeats SCI slave reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remarks 3. When performing a SCI slave transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Slave_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_IIC_StartCondition

Sends the start bit.

Remark This API function is called as the internal function of R_SCIn_IIC_Master_Send and R_SCIn_IIC_Master_Receive.

[Syntax]

```
void    R_SCIn_IIC_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_IIC_StopCondition

Sends the stop bit.

[Syntax]

```
void    R_SCIn_IIC_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scin_transmitend_interrupt callback routine.

[Syntax]

```
static  void    r_scin_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveend

Performs processing in response to the receive-data-full interrupts.

Remark This API function is called as the r_scin_receive_interrupt callback routine.

[Syntax]

```
static  void    r_scin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveerror

Performs processing in response to the receive error interrupts.

Remark This API function is called as the r_scin_receiveerror_interrupt callback routine.

[Syntax]

```
static void r_scin_callback_receiveerror ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.26 FIFO embedded serial communications interface (SCIFA)

Below is a list of API functions output by the Code Generator for FIFO embedded serial communications interface use.

Table 3.26 API Functions: [FIFO Embedded Serial Communications Interface]

API Function Name	Function
R_SCIFAn_Create	Performs initialization necessary to control the FIFO embedded serial communications interface.
R_SCIFAn_Create_UserInit	Performs user-defined initialization relating to the FIFO embedded serial communications interface.
r_scifan_teif_interrupt	Performs processing in response to the transmit-end interrupts.
r_scifan_txif_interrupt	Performs processing in response to the transmit FIFO data empty interrupts.
r_scifan_rxif_interrupt	Performs processing in response to the receive FIFO data full interrupts.
r_scifan_erif_interrupt	Performs processing in response to the framing error or parity error interrupts.
r_scifan_brif_interrupt	Performs processing in response to the break or overrun interrupts.
r_scifan_drif_interrupt	Performs processing in response to the receive data ready interrupts.
r_scifan_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scifan_callback_receiveend	Performs processing in response to the receive FIFO data full interrupts.
r_scifan_callback_error	Performs processing in response to the error interrupts.
R_SCIFAn_Start	Starts FIFO embedded SCI communication.
R_SCIFAn_Stop	Ends FIFO embedded SCI communication.
R_SCIFAn_Serial_Send	EStarts FIFO embedded SCI transmission (asynchronous mode).
R_SCIFAn_Serial_Receive	Starts FIFO embedded SCI reception (asynchronous mode).
R_SCIFAn_Serial_Send_Receive	Starts FIFO embedded SCI transmission/reception (clock synchronous mode).

R_SCIFA*n*_Create

Performs initialization necessary to control the FIFO embedded serial communications interface.

[Syntax]

```
void    R_SCIFAn_Create ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Create_UserInit

Performs user-defined initialization related to the FIFO embedded serial communications interface.

Remark This API function is called as the R_SCIFAn_Create callback routine.

[Syntax]

```
void    R_SCIFAn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_teif_interrupt

Performs processing in response to the transmit-end interrupts.

Remark This API function is called to run the interrupt processing for the transmit-end interrupt.

[Syntax]

```
static  void    r_scifan_teif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_txif_interrupt

Performs processing in response to the transmit FIFO data empty interrupts.

Remark This API function is called to run the interrupt processing for the transmit FIFO data empty interrupt.

[Syntax]

```
static  void    r_scifan_txif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_rxif_interrupt

Performs processing in response to the receive FIFO data full interrupts.

Remark This API function is called to run the interrupt processing for the receive FIFO data full interrupt.

[Syntax]

```
static  void    r_scifan_rxif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_erif_interrupt

Performs processing in response to the framing error or parity error interrupts.

Remark This API function is called to run the interrupt processing for the framing error or parity error interrupt.

[Syntax]

```
static  void    r_scifan_erif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_brif_interrupt

Performs processing in response to the break or overrun interrupts.

Remark This API function is called to run the interrupt processing for the break or overrun interrupt.

[Syntax]

```
static  void    r_scifan_brif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_drif_interrupt

Performs processing in response to the receive data ready interrupts.

Remark This API function is called to run the interrupt processing for the receive data ready interrupt.

[Syntax]

```
static  void    r_scifan_drif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scifan_teif_interrupt callback routine.

[Syntax]

```
static  void    r_scifan_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_receiveend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scifan_txif_interrupt callback routine.

[Syntax]

```
static  void    r_scifan_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_error

Performs processing in response to the error interrupts.

Remark This API function is called as the r_scifan_erif_interrupt or r_scifan_brif_interrupt callback routine.

[Syntax]

```
static  void    r_scifan_callback_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Start

Starts FIFO embedded SCI communication.

[Syntax]

```
void    R_SCIFAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Stop

Ends FIFO embedded SCI communication.

[Syntax]

```
void    R_SCIFAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Serial_Send

Starts FIFO embedded SCI transmission (asynchronous mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a SCI transmission, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
MD_STATUS   R_SCIFAn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Receive

Starts FIFO embedded SCI reception (asynchronous mode).

Remarks 1. This API function repeats the byte-level SCI reception from the buffer specified in argument *rx_buf* the number of times specified in argument *rx_num*.

Remarks 2. When performing a SCI reception, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
MD_STATUS   R_SCIFAn_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t         rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Send_Receive

Starts FIFO embedded SCI transmission/reception (clock synchronous mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. This API function repeats the byte-level SCI reception from the buffer specified in argument *rx_buf* the number of times specified in argument *rx_num*.

Remarks 3. When performing a SCI transmission/reception, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
MD_STATUS   R_SCIn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

3.2.27 I²C bus interface (RIIC)

Below is a list of API functions output by the Code Generator for I^2C bus interface use.

Table 3.27 API Functions: [I^2C Bus Interface]

API Function Name	Function
R_RIICn_Create	Performs initialization necessary to control the I^2C bus interface.
R_RIICn_Create_UserInit	Performs user-defined initialization relating to the I^2C bus interface.
r_riicn_error_interrupt	Performs processing in response to the transfer error/event generation interrupts (EEI).
r_riicn_receive_interrupt	Performs processing in response to the receive data full interrupts (RXI).
r_riicn_transmit_interrupt	Performs processing in response to the transmit data empty interrupts (TXI).
r_riicn_transmitend_interrupt	Performs processing in response to the transmit end interrupts (TEI).
R_RIICn_Start	Starts RIIC communication.
R_RIICn_Stop	Ends RIIC communication.
R_RIICn_Master_Send	Starts RIIC master transmission.
R_RIICn_Master_Receive	Starts RIIC master reception.
R_RIICn_Slave_Send	Starts RIIC slave transmission.
R_RIICn_Slave_Receive	Starts RIIC slave reception.
R_RIICn_StartCondition	Issues the start condition and causes a transfer error and an event generation interrupt (EEI).
R_RIICn_StopCondition	Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).
r_riicn_callback_receiveerror	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.
r_riicn_callback_transmitend	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.
r_riicn_callback_receiveend	Of the interrupt processing for transfer error/event generation interrupts (EEI), processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive is performed.

R_RIIC*n*_Create

Performs initialization necessary to control the I^{2}C bus interface.

[Syntax]

```
void    R_RIICn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Create_UserInit

Performs user-defined initialization relating to the I²C bus interface.

Remark This API function is called as the R_RIICn_Create callback routine.

[Syntax]

```
void    R_RIICn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_error_interrupt

Performs processing in response to the transfer error/event generation interrupts (EEI).

Remark This API function is called to run interrupt processing for the transfer error/event generation interrupts (EEI), which are generated when the I^2C bus interface detects the transfer error/event generation (arbitration-lost, NACK, timeout, start condition, and stop condition).

[Syntax]

```
static  void    r_riicn_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_receive_interrupt

Performs processing in response to the receive data full interrupts (RXI).

Remark This API function is called to run interrupt processing for the receive data full interrupts (RXI).

[Syntax]

```
static  void    r_riicn_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmit_interrupt

Performs processing in response to the transmit data empty interrupts (TXI).

Remark This function is called to run interrupt processing for the transmit data empty interrupts (TXI).

[Syntax]

```
static  void    r_riicn_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmitend_interrupt

Performs processing in response to the transmit end interrupts (TEI).

Remark This API function is called to run interrupt processing for the transmit end interrupts (TEI).

[Syntax]

```
static  void    r_riicn_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Start

Starts RIIC communication.

[Syntax]

```
void    R_RIICn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Stop

Ends RIIC communication.

[Syntax]

```
void    R_RIICn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIICn_Master_Send

Starts RIIC master transmission.

Remarks 1. This API function handles RIIC master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. RIIC master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master transmission.

Remarks 3. When performing a RIIC master transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RIICn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num
);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Master_Receive

Starts RIIC master reception.

Remarks 1. This API function handles RIIC master transmission to the slave device at the slave address specified by the argument *adr*. RIIC master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remarks 2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master reception.

Remarks 3. When performing a RIIC master reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RIICn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t
rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Slave_Send

Starts RIIC slave transmission.

Remarks 1. This API function repeats the byte-level RIIC slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a RIIC slave transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

None.

R_RIICn_Slave_Receive

Starts RIIC slave reception.

Remarks 1. This API function performs byte-level RIIC slave reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remarks 2. When performing a RIIC slave reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t         rx_num;`	Total amount of data to receive

[Return value]

None.

R_RIICn_StartCondition

Issues the start condition and causes a transfer error and an event generation interrupt (EEI).

Remarks 1. This API function is called as the internal function of R_RIICn_Master_Send and R_RIICn_Master_Receive.

Remarks 2. r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void    R_RIICn_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_StopCondition

Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).

Remark r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void    R_RIICn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveerror

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_riicn_callback_receiveerror ( MD_STATUS status );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS status;`	Source of the transfer errors and event generation interrupts MD_ERROR1: Arbitration-lost detection MD_ERROR2: Timeout detection MD_ERROR3: NACK detection

[Return value]

None.

r_riic*n*_callback_transmitend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static  void    r_riicn_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static  void    r_riicn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.28 Serial peripheral interface (RSPI)

Below is a list of API functions output by the Code Generator for serial peripheral interface use.

Table 3.28 API Functions: [Serial Peripheral Interface]

API Function Name	Function
R_RSPIn_Create	Performs initialization necessary to control the serial peripheral interface.
R_RSPIn_Create_UserInit	Performs user-defined initialization relating to the serial peripheral interface.
r_rspin_receive_interrupt	Performs processing in response to the receive buffer full interrupts.
r_rspin_transmit_interrupt	Performs processing in response to the transmit buffer error interrupts.
r_rspin_error_interrupt	Performs processing in response to the RSPI error interrupts.
r_rspin_idle_interrupt	Performs processing in response to the RSPI idle interrupts.
R_RSPIn_Start	Starts RSPI communication.
R_RSPIn_Stop	Ends RSPI communication.
R_RSPIn_Send	Starts RSPI transmission.
R_RSPIn_Send_Receive	Starts RSPI transmission/reception.
r_rspin_callback_receiveend	Performs processing in response to the receive buffer full interrupts.
r_rspin_callback_error	Performs processing in response to the RSPI error interrupts.
r_rspin_callback_transmitend	Performs processing in response to the RSPI idle interrupts.

R_RSPI*n*_Create

Performs initialization necessary to control the serial peripheral interface.

[Syntax]

```
void    R_RSPIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Create_UserInit

Performs user-defined initialization relating to the serial peripheral interface.

Remark This API function is called as the R_RSPIn_Create callback routine.

[Syntax]

```
void    R_RSPIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_receive_interrupt

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called to run interrupt processing for the receive buffer full interrupt.

[Syntax]

```
static  void    r_rspin_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_transmit_interrupt

Performs processing in response to the transmit buffer empty interrupts.

Remark This API function is called to run interrupt processing for the transmit buffer empty interrupts.

[Syntax]

```
static  void    r_rspin_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_error_interrupt

Performs processing in response to the RSPI error interrupts.

Remark This API function is called to run interrupt processing for the RSPI error interrupts.

[Syntax]

```
static  void    r_rspin_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_idle_interrupt

Performs processing in response to the RSPI idle interrupts.

Remark This API function is called to run interrupt processing for the RSPI idle interrupts.

[Syntax]

```
static  void    r_rspin_idle_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Start

Starts RSPI communication.

[Syntax]

```
void    R_RSPIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Stop

Ends RSPI communication.

[Syntax]

```
void    R_RSPIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Send

Starts RSPI transmission.

Remarks 1. This API function repeats the byte-level RSPI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remarks 2. When performing a RSPI transmission, R_RSPIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RSPIn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t        tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_RSPI*n*_Send_Receive

Starts RSPI transmission/reception.

Remarks 1. This API function repeats RSPI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remarks 2. This API function repeats RSPI reception processing in byte units the number of times specified by the argument *tx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remarks 3. When performing a RSPI transmission/reception, R_RSPIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RSPIn_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num, uint8_t *
const rx_buf );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t        tx_num;`	Total amount of data to send/receive
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

r_rspi*n*_callback_receiveend

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called as the r_rspin_receive_interrupt callback routine.

[Syntax]

```
static  void    r_rspin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_callback_error

Performs processing in response to the RSPI error interrupts.

Remark This API function is called as the r_rspin_error_interrupt callback routine.

[Syntax]

```
static  void    r_rspin_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Source of the RSPI error interrupt (x is undefined) xxxx00x1B: Overrun error detection xxxx01x0B: Mode fault error detection xxxx10x0B: Parity error detection

[Return value]

None.

r_rspi*n*_callback_transmitend

Performs processing in response to the RSPI idle interrupts.

Remark This API function is called as the r_rspin_idle_interrupt callback routine.

[Syntax]

```
static  void    r_rspin_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.29 CRC calculator (CRC)

Below is a list of API functions output by the Code Generator for CRC calculator use.

Table 3.29 API Functions: [CRC calculator]

API Function Name	Function
R_CRC_SetCRC8	Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).
R_CRC_SetCRC16	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).
R_CRC_SetCCITT	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).
R_CRC_SetCRC32	Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$)
R_CRC_SetCRC32C	Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$)
R_CRC_Input_Data	Sets the initial value of the data from which the CRC is to be calculated.
R_CRC_Get_Result	Gets the result of operation.

R_CRC_SetCRC8

Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCRC8 ( void );
```

Other devices

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC8 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder` *`order`*`;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_SetCRC16

Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCRC16 ( void );
```

Other devices

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC16 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder order;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_SetCCITT

Initializes the CRC calculator for the 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCCITT ( void );
```

Other devices

```
#include    "r_cg_crc.h"
void    R_CRC_SetCCITT ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder order;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_SetCRC32

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X + 1$).

[Syntax]

```
void    R_CRC_SetCRC32 ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_SetCRC32C

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$).

[Syntax]

```
void    R_CRC_SetCRC32C ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_Input_Data

Sets the initial value of the data from which the CRC is to be calculated.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_CRC_Input_Data ( uint8_t data );
```

[Argument(s)]

I/O	Argument	Description
I	uint8_t *data*;	The initial value of the data from which the CRC is to be calculated

[Return value]

None.

R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```
#include    "r_cg_macrodriver"
void    R_CRC_Get_Result ( uint8_t * const result );
```

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const result;`	Pointer to the location where the result of operation is stored

[Return value]

None.

3.2.30 12-bit A/D converter (S12AD)

Below is a list of API functions output by the Code Generator for 12-bit A/D converter use.

Table 3.30 API Functions: [12-Bit A/D Converter]

API Function Name	Function
R_S12ADn_Create	Performs initialization necessary to control the 12-bit A/D converter.
R_S12ADn_Create_UserInit	Performs user-defined initialization relating to the 12-bit A/D converter.
r_s12adn_interrupt	Performs processing in response to the A/D scan end interrupt.
r_s12adn_groupb_interrupt	Performs processing in response to the group B scan end interrupt.
R_S12ADn_Start	Starts A/D conversion.
R_S12ADn_Stop	Ends A/D conversion.
R_S12ADn_Get_ValueResult	Gets the result of conversion.
R_S12ADn_Set_CompareValue	Sets compare level.
r_s12adn_compare_interrupt	Performs processing in response to the compare interrupt.

R_S12AD*n*_Create

Performs initialization necessary to control the 12-bit A/D converter.

[Syntax]

```
void    R_S12ADn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Create_UserInit

Performs user-defined initialization relating to the 12-bit A/D converter.

Remark This API function is called as the R_S12ADn_Create callback routine.

[Syntax]

```
void    R_S12ADn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad*n*_interrupt

Performs processing in response to the A/D scan end interrupt.

Remark This API function is called to run interrupt processing for the A/D scan end interrupt, which is generated on completion of scanning of the analog inputs.

[Syntax]

```
static  void    r_s12adn_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad*n*_groupb_interrupt

Performs processing in response to the group B scan end interrupt.

Remark This function is called to run interrupt processing for the group B scan end interrupt, which is generated when scanning of the analog inputs allocated to group B is completed.

[Syntax]

```
static  void    r_s12adn_groupb_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Start

Starts A/D conversion.

[Syntax]

```
void    R_S12ADn_Start ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Stop

Ends A/D conversion.

[Syntax]

```
void    R_S12ADn_Stop ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Get_ValueResult

Gets the result of conversion.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_s12ad.h"
void    R_S12ADn_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	`ad_channel_t channel;`	Channel number ADCHANNEL0: Input channel AN000 ADCHANNEL1: Input channel AN001 ADCHANNEL2: Input channel AN002 ADCHANNEL3: Input channel AN003 ADCHANNEL4: Input channel AN004 ADCHANNEL6: Input channel AN006 ADCHANNEL8: Input channel AN008 ADCHANNEL9: Input channel AN009 ADCHANNEL10: Input channel AN010 ADCHANNEL11: Input channel AN011 ADCHANNEL12: Input channel AN012 ADCHANNEL13: Input channel AN013 ADCHANNEL14: Input channel AN014 ADCHANNEL15: Input channel AN015 ADTEMPSENSOR: Extended analog input (temperature sensor output) ADINTERREFVOLT: Extended analog input (internal reference voltage)
O	`uint16_t * const buffer;`	Pointer to the area where the results of conversion are stored

[Return value]

None.

R_S12AD*n*_Set_CompareValue

Sets compare level.

[Syntax]

```
void    R_S12ADn_Set_CompareValue ( ad_channel_t  reg_value0,  rad_channel_t
reg_value1 );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	`ad_chanel_t` *`reg_value0`*	Register value set to the compare revel register 0
I	`ad_chanel_t` *`reg_value1`*	Register value set to the compare revel register 1

[Return value]

None.

r_s12ad*n*_compare_interrupt

Performs processing in response to the compare interrupt.

Remark This API function is called to run the interrupt processing for the compare interrupt.

[Syntax]

```
void    r_s12adn_compare_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.31 D/A converter (DA)

Below is a list of API functions output by the Code Generator for D/A converter use.

Table 3.31 API Functions: [D/A Converter]

API Function Name	Function
R_DA_Create	Performs initialization necessary to control the D/A converter.
R_DA_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DAm_Start	Starts D/A conversion.
R_DAm_Stop	Ends D/A conversion.
R_DAm_Set_ConversionValue	Sets the data for D/A conversion.

R_DA_Create

Performs initialization necessary to control the D/A converter.

[Syntax]

```
void    R_DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DA_Create_UserInit

Performs user-defined initialization relating to the D/A converter.

Remark This API function is called as the R_DA_Create callback routine.

[Syntax]

```
void    R_DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Start

Starts D/A conversion.

[Syntax]

```
void    R_DAm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Stop

Ends D/A conversion.

[Syntax]

```
void    R_DAm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Set_ConversionValue

Sets the data for D/A conversion.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DAm_Set_ConversionValue ( uint16_t reg_value );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t reg_value;`	Data for D/A conversion

[Return value]

None.

3.2.32 12-bit converter (R12DA)

Below is a list of API functions output by the Code Generator for 12-bit D/A converter use.

Table 3.32 API Functions: [12-Bit D/A Converter]

API Function Name	Function
R_R12DA_Create	Performs initialization necessary to control the 12-bit D/A converter.
R_DA_Create_UserInit	Performs user-defined initialization relating to the 12-bit D/A converter.
R_R12DAn_Start	Starts D/A conversion.
R_R12DAn_Stop	Ends D/A conversion.
R_R12DAn_Start	Starts synchronous D/A conversion.
R_R12DAn_Stop	Ends synchronous D/A conversion.
R_R12DAn_Set_ConversionValue	Sets the data for D/A conversion.

R_R12DA_Create

Performs initialization necessary to control the 12-bit D/A converter.

[Syntax]

```
void    R_R12DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DA_Create_UserInit

Performs user-defined initialization relating to the 12-bit D/A converter.

Remark This API function is called as the R_R12DA_Create callback routine.

[Syntax]

```
void    R_R12DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Start

Starts 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Stop

Ends 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_R12DA_sync_Start

Starts synchronous 12-bit D/A conversion.

[Syntax]

```
void    R_R12DA_sync_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA_sync_Stop

Ends synchronous 12-bit D/A conversion.

[Syntax]

```
void    R_R12DA_sync_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Set_ConversionValue

Sets the data for 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Set_ConversionValue ( uint16_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t reg_value;`	Data for 12-bit D/A conversion

[Return value]

None.

3.2.33 Comparator B (CMPB)

Below is a list of API functions output by the Code Generator for Comparator B use.

Table 3.33 API Functions: [Comparator B]

API Function Name	Function
R_CMPB_Create	Performs initialization necessary to control the Comparator B.
R_CMPB_Create_UserInit	Performs user-defined initialization relating to the Comparator B.
r_cmpb_cmpbn_interrupt	Performs processing in response to the comparator B interrupt.
R_CMPBn_Start	Starts comparison for analog input voltage.
R_CMPBn_Stop	Ends comparison for analog input voltage.

R_CMPB_Create

Performs initialization necessary to control the Comparator B.

[Syntax]

```
void    R_CMPB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CMPB_Create_UserInit

Performs user-defined initialization relating to the Comparator B.

Remark This API function is called as the R_CMPB_Create callback routine.

[Syntax]

```
void    R_CMPB_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cmpb_cmpb*n*_interrupt

Performs processing in response to the comparator B interrupt.

Remark This API function is called to run interrupt processing for the comparator B*n* interrupt, which is generated when the comparison result changes at this time.

[Syntax]

```
void    r_cmpb_cmpbn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPB*n*_Start

Starts comparison for analog input voltage.

[Syntax]

```
void    R_CMPBn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPB*n*_Stop

Ends comparison for analog input voltage.

[Syntax]

```
void    R_CMPBn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.34 Data operation circuit (DOC)

Below is a list of API functions output by the Code Generator for data operation circuit.

Table 3.34 API Functions: [Data Operation Circuit]

API Function Name	Function
R_DOC_Create	Performs initialization necessary to control the data operation circuit.
R_DOC_Create_UserInit	Performs user-defined initialization relating to the data operation circuit.
r_doc_dopcf_interrupt	Performs processing in response to the data operation circuit interrupt.
R_DOC_SetMode	Sets the operating mode and the initial value of the reference value for use by the data operation circuit.
R_DOC_WriteData	Sets the input value (value for comparison with, addition to, or subtraction from the reference value) for use by the data operation circuit.
R_DOC_GetResult	Gets the result of operation.
R_DOC_ClearFlag	Clears the data operation circuit flag.

R_DOC_Create

Performs initialization necessary to control the data operation circuit.

[Syntax]

```
void    R_DOC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Create_UserInit

Performs user-defined initialization relating to the data operation circuit.

Remark This API function is called as the R_DOC_Create callback routine.

[Syntax]

```
void    R_DOC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_doc_dopcf_interrupt

Performs processing in response to the data operation circuit interrupt.

Remark This API function is called to run interrupt processing for the data operation circuit interrupt, which is generated when the result of data comparison satisfies the condition for detection, the result of addition is greater than 0xFFFF, or the result of subtraction is less than 0x00.

[Syntax]

```
static  void    r_doc_dopcf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

Sets the operating mode and the initial value of the reference value for use by the data operation circuit.

Remarks 1. When COMPARE_MISMATCH or COMPARE_MATCH (data comparison mode) is specified as the mode of operation, the 16-bit reference value is stored in the DOC data setting register (DODSR).

Remarks 2. When ADDITION (data addition mode) or SUBTRACTION (data subtraction mode) is specified for the mode (operation mode), the 16-bit value is stored in the DOC data setting register (DODSR) as the initial value.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_doc.h"
void    R_DOC_SetMode ( doc_mode_t mode, uint16_t value );
```

[Argument(s)]

I/O	Argument	Description
I	doc_mode_t *mode*;	Operating modes (including the condition for detection) COMPARE_MISMATCH: Data comparison mode (mismatch) COMPARE_MATCH: Data comparison mode (match) ADDITION: Data addition mode SUBTRACTION: Data subtraction mode
I	uint16_t *value*;	Initial value of the reference value for use by the DOC

[Return value]

None.

R_DOC_WriteData

Sets the value for comparison with, addition to, or subtraction from the reference value.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DOC_WriteData ( uint16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	`uint16_t data;`	Input data for use in operation

[Return value]

None.

R_DOC_GetResult

Gets the result of operation.

[Syntax]

```
#include    "r_cg_macrodriver"
void    R_DOC_GetResult ( uint16_t * const data );
```

[Argument(s)]

I/O	Argument	Description
O	`uint16_t * const data;`	Pointer to the location where the result of operation is to be stored

[Return value]

None.

R_DOC_ClearFlag

Clears the data operation circuit flag.

[Syntax]

```
void    R_DOC_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.35 Low power timer (LPT)

Below is a list of API functions output by the Code Generator for low power timer use.

Table 3.35 API Functions: [Low power timer]

API Function Name	Function
R_LPT_Create	Performs initialization necessary to control the low power timer.
R_LPT_Create_UserInit	Performs user-defined initialization relating to the low power timer.
R_LPT_Start	Starts counting by a low power timer.
R_LPT_Stop	Ends counting by a low power timer.

R_LPT_Create

Performs initialization necessary to control the low power timer.

[Syntax]

```
void    R_LPT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Create_UserInit

Performs user-defined initialization relating to the low power timer.

Remark This API function is called as the R_LPT_Create callback routine.

[Syntax]

```
void    R_LPT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Start

Starts counting by a low power timer.

[Syntax]

```
void    R_LPT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Stop

Ends counting by a low power timer.

[Syntax]

```
void    R_LPT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.36 Comparator C (CMPC)

Below is a list of API functions output by the Code Generator for Comparator B use.

Table 3.36 API Functions: [Comparator B]

API Function Name	Function
R_CMPC_Create	Performs initialization necessary to control the Comparator C.
R_CMPC_Create_UserInit	Performs user-defined initialization relating to the Comparator C.
r_cmpc_cmpcn_interrupt	Performs processing in response to the comparator C interrupt.
R_CMPCn_Start	Starts comparison for analog input voltage.
R_CMPCn_Stop	Ends comparison for analog input voltage.

R_CMPC_Create

Performs initialization necessary to control the Comparator C.

[Syntax]

```
void    R_CMPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CMPC_Create_UserInit

Performs user-defined initialization relating to the Comparator C.

Remark This API function is called as the R_CMPC_Create callback routine.

[Syntax]

```
void    R_CMPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cmpc_cmpc*n*_interrupt

Performs processing in response to the comparator C interrupt.

Remark This API function is called to run interrupt processing for the comparator C*n* interrupt, which is generated when the comparison result changes at this time.

[Syntax]

```
void    r_cmpc_cmpcn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPC*n*_Start

Starts comparison for analog input voltage.

[Syntax]

```
void    R_CMPCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPC*n*_Stop

Ends comparison for analog input voltage.

[Syntax]

```
void    R_CMPCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.37 LCD controller / driver (LCD)

Below is a list of API functions output by the Code Generator for LCD controller / driver use.

Table 3.37 API Functions: [LCD controller / driver]

API Function Name	Function
R_LCD_Create	Performs initialization necessary to control the LCD controller / driver.
R_LCD_Create_UserInit	Performs user-defined initialization relating to the LCD controller / driver.
R_LCD_Start	Sets the LCD controller / driver to display on status.
R_LCD_Stop	Sets the LCD controller / driver to display off status.
R_LCD_Voltage_On	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Voltage_Off	Disables operation of internal voltage boost circuit and capacitor split circuit.

R_LCD_Create

Performs initialization necessary to control the LCD controller / driver.

[Syntax]

```
void    R_LCD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Create_UserInit

Performs user-defined initialization relating to the LCD controller / driver.

Remark This API function is called as the R_LCD_Create callback routine.

[Syntax]

```
void    R_LCD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Start

Sets the LCD controller / driver to display on status.

[Syntax]

```
void    R_LCD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Stop

Sets the LCD controller / driver to display off status.

[Syntax]

```
void    R_LCD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Voltage_On

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Voltage_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Voltage_Off

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Voltage_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug 01, 2014	-	First Edition issued
1.10	Jun 27, 2014	All	RX64M supported
1.20	Aug 01, 2014	All	Comparator B supported
		All	R_LPC_AllModuleStop → R_LPC_AllModuleClockStop
		3.2.1 Common	
		19	Remark changed.
		23	Remark changed
		3.2.21 Compare match timer W (CMTW)	
		175, 180	output capture interrupt → output compare interrupt
		3.2.22 Realtime clock (RTC)	
		183	Realtime clock (RTCA) → Realtime clock (RTC)
1.30	Oct 01, 2016	All	RX65N / RX651 supported
		All	Low power timer supported
		All	Comparator C supported
		All	LCD controller / driver supported
		3.2.1 Common	
		22	PowerON_Reset_PC added
		23	r_privileged_exception added
		24	r_floatingpoint_exception added
		25	r_access_exception added
		29	Remark changed
		3.2.4 lock frequency accuracy measurement circuit (CAC)	
		51	r_cac_ovrf_interrupt → r_cac_ovff_interrupt
		3.2.9 Data transfer controller (DTC)	
		99	Remark changed
		100	Remark changed
		3.2.10 Event link controller (ELC)	
		104	Remark changed
		3.2.12 Multi-function timer pulse unit 2 (MTU2)	
		117	r_mtu2_cj_tgimn_interrupt added
		119	r_mtu2_cj_tcivn_interrupt added

Rev.	Date	Description	
		Page	Summary
1.30	Oct 01, 2016	3.2.13 Multi-function timer pulse unit 3 (MTU3)	
		127	r_mtu3_cj_tgimn_interrupt added
		129	r_mtu3_cj_tcivn_interrupt added
		3.2.14 Port output enable 2 (POE2)	
		139	R_POE2_Set_HiZ_MTUn added
		140	R_POE2_Clear_HiZ_MTUn added
		3.2.15 Port output enable 3 (POE3)	
		147	R_POE3_Set_HiZ_MTUn added
		148	R_POE3_Clear_HiZ_MTUn added
		149	R_POE3_Set_HiZ_GPTn added
		150	R_POE3_Clear_HiZ_GPTn added
		3.2.23 Watchdog timer (WDT)	
		224	r_wdt_nmi_interrupt added
		3.2.29 CRC Calculator (CRC)	
		311	R_CRC_SetCRC32 added
		312	R_CRC_SetCRC32C added

CS+ Code Generator Tool User's Manual:
RX API Reference

Publication Date:	Rev.1.00	Aug 01, 2014
	Rev.1.30	Oct 01, 2016

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

CS+ Code Generator Tool

Renesas Electronics Corporation

R20UT3103EJ0130