User's Manual

e² studio Code Generator

Integrated Development Environment

User's Manual: RL78 API Reference

Target Device

RL78 Family

Renesas Electronics
www.renesas.com

Rev.1.05 Feb 2018

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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

How to Use This Manual

Readers

The target readers of this manual are the application system engineers who use the Code Generator and need to understand its function.

Purpose

The purpose of this manual is to explain the user for understanding and using the Code Generator functions.
We aim to help their system development including their hardware and software.

Organization

This manual can be broadly divided into the following units.

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:	$\overline{\text{XXX}}$ (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

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	Output (*1)
Common	r_main.c or r_cg_main.c	main R_MAIN_UserInit	A A
	r_systeminit.c or r_cg_systeminit.c	hdwinit R_Systeminit	A A
	r_cg_macrodriver.h	-	-
	r_cg_userdefine.h	-	-
Clock generator	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_RAMECC_Start R_CGC_RAMECC_Stop R_CGC_StackPointer_Start R_CGC_StackPointer_Stop R_CGC_ClockMonitor_Start R_CGC_ClockMonitor_Stop	A M A A A A A A
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_ram_ecc_interrupt r_cgc_stackpointer_interrupt r_cgc_clockmonitor_interrupt R_CGC_Get_ResetSource	M A A A A
	r_cg_cgc.h	-	-
Port functions	r_cg_port.c	R_PORT_Create	A
	r_cg_port_user.c	R_PORT_Create_UserInit	M
	r_cg_port.h	-	-
High-speed on-chip Oscillator clock Frequency Correction function	r_cg_hofc.c	R_HOFC_Create R_HOFC_Start R_HOFC_Stop	A A A
	r_cg_hofc_user.c	R_HOFC_Create_UserInit r_hofc_interrupt	M A
	r_cg_hofc.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Timer array unit	r_cg_timer.c or r_cg_tau.c	R_TAUm_Create R_TAUm_Channeln_Start R_TAUm_Channeln_Higher8bits_Start R_TAUm_Channeln_Lower8bits_Start R_TAUm_Channeln_Stop R_TAUm_Channeln_Higher8bits_Stop R_TAUm_Channeln_Lower8bits_Stop R_TAUm_Reset R_TAUm_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_Set_SoftwareTriggerOn	A A A A A A A M A A A
	r_cg_timer_user.c or r_cg_tau_user.c	R_TAUm_Create_UserInit r_taum_channeln_interrupt r_taum_channeln_higher8bits_interrupt	M A A
	r_cg_timer.h or r_cg_tau.h	-	-
Timer RJ	r_cg_timer.c or r_cg_tmrj.c	R_TMR_RJn_Create R_TMR_RJn_Start R_TMR_RJn_Stop R_TMR_RJn_Set_PowerOff R_TMR_RJn_Get_PulseWidth R_TMRJn_Create R_TMRJn_Start R_TMRJn_Stop R_TMRJn_Set_PowerOff R_TMRJn_Get_PulseWidth	A A A A A A A A M M
	r_cg_timer_user.c or r_cg_tmrj_user.c	R_TMR_RJn_Create_UserInit r_tmr_rjn_interrupt R_TMRJn_Create_UserInit r_tmrjn_interrupt	M A M A
	r_cg_timer.h or r_cg_tmrj.h	-	-
Timer RD	r_cg_timer.c or r_cg_tmrd.c	R_TMR_RDn_Create R_TMR_RDn_Start R_TMR_RDn_Stop R_TMR_RDn_Set_PowerOff R_TMR_RDn_ForcedOutput_Start R_TMR_RDn_ForcedOutput_Stop R_TMR_RDn_Get_PulseWidth R_TMRDn_Create R_TMRDn_Start R_TMRDn_Stop R_TMRDn_Set_PowerOff R_TMRDn_ForcedOutput_Start R_TMRDn_ForcedOutput_Stop R_TMRDn_Get_PulseWidth R_TMRD_Set_PowerOff	A A A M M M A A A A M M M M
	r_cg_timer_user.c or r_cg_tmrd_user.c	R_TMR_RDn_Create_UserInit r_tmr_rdn_interrupt R_TMRDn_Create_UserInit r_tmrdn_interrupt	M A M A
	r_cg_timer.h or r_cg_tmrd.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Timer RG	r_cg_timer.c	R_TMR_RG0_Create R_TMR_RG0_Start R_TMR_RG0_Stop R_TMR_RG0_Set_PowerOff R_TMR_RG0_Get_PulseWidth	A A A M A
	r_cg_timer_user.c	R_TMR_RG0_Create_UserInit r_tmr_rg0_interrupt	M A
	r_cg_timer.h	-	-
Timer RX	r_cg_tmrx.c	R_TMRX_Create R_TMRX_Start R_TMRX_Stop R_TMRX_Set_PowerOff R_TMRX_Get_BufferValue	A A A A M
	r_cg_tmrx_user.c	R_TMRX_Create_UserInit r_tmrx_interrupt	M A
	r_cg_tmrx.h	-	-
16-bit timer KB	r_cg_timer.c or r_cg_tmkb.c	R_TMR_KB_Create R_TMR_KBm_Start R_TMR_KBm_Stop R_TMR_KBm_Set_PowerOff R_TMR_KBmn_ForcedOutput_Start R_TMR_KBmn_ForcedOutput_Stop R_TMR_KBm_BatchOverwriteRequestOn R_TMR_KBm_ForcedOutput_mn_Start R_TMR_KBm_ForcedOutput_mn_Stop R_TMR_KBm_Reset	A A A M A A A A A M
	r_cg_timer_user.c or r_cg_tmkb_user.c	R_TMR_KBm_Create_UserInit r_tmr_kbm_interrupt	M A
	r_cg_timer.h or r_cg_tmkb.h	-	-
16-bit timer KC0	r_cg_timer.c	R_TMR_KC0_Create R_TMR_KC0_Start R_TMR_KC0_Stop R_TMR_KC0_Set_PowerOff	A A A M
	r_cg_timer_user.c	R_TMR_KC0_Create_UserInit r_tmr_kc0_interrupt	M A
	r_cg_timer.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
16-bit timer KB2	r_cg_kb2.c	R_KB2m_Create	A
		R_KB2m_Start	A
		R_KB2m_Stop	A
		R_KB2m_Set_PowerOff	M
		R_KB2m_Simultaneous_Start	A
		R_KB2m_Simultaneous_Stop	A
		R_KB2m_Synchronous_Start	A
		R_KB2m_Synchronous_Stop	A
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	A
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	A
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	A
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	A
		R_KB2m_TKBOn0_DitheringFunction_Start	A
		R_KB2m_TKBOn0_DitheringFunction_Stop	A
		R_KB2m_TKBOn1_DitheringFunction_Start	A
		R_KB2m_TKBOn1_DitheringFunction_Stop	A
		R_KB2m_TKBOn0_SmoothStartFunction_Start	A
		R_KB2m_TKBOn0_SmoothStartFunction_Stop	A
		R_KB2m_TKBOn1_SmoothStartFunction_Start	A
		R_KB2m_TKBOn1_SmoothStartFunction_Stop	A
		R_KB2m_Set_BatchOverwriteRequestOn	A
	r_cg_kb2_user.c	R_KB2m_Create_UserInit	M
		r_kb2m_interrupt	A
	r_cg_kb2.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Real-time clock	r_cg_rtc.c	R_RTC_Create R_RTC_Start R_RTC_Stop R_RTC_Set_PowerOff R_RTC_Set_HourSystem R_RTC_Set_CounterValue R_RTC_Set_CalendarCounterValue R_RTC_Set_BinaryCounterValue R_RTC_Get_CounterValue R_RTC_Get_CalendarCounterValue R_RTC_Get_BinaryCounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_CalendarAlarmOn R_RTC_Set_BinaryAlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Set_CalendarAlarmValue R_RTC_Set_BinaryAlarmValue R_RTC_Get_AlarmValue R_RTC_Get_CalendarAlarmValue R_RTC_Get_BinaryAlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff R_RTC_Set_RTCOUTOn R_RTC_Set_RTCOUTOff	A A A A M A A A A A A A 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_interrupt r_rtc_callback_constperiod r_rtc_callback_alarm r_rtc_alarminterrupt r_rtc_periodicinterrupt r_rtc_callback_periodic	M A A A A A A
	r_cg_rtc.h	-	-
Subsystem clock frequency measure-ment circuit	r_cg_fmc.c	R_FMC_Create R_FMC_Start R_FMC_Stop R_FMC_Set_PowerOff	A A A M
	r_cg_fmc_user.c	R_FMC_Create_UserInit r_fmc_interrupt	M A
	r_cg_fmc.h	-	-
12-bit interval timer	r_cg_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Reset R_IT_Set_PowerOff	A A A M M
	r_cg_it_user.c	R_IT_Create_UserInit r_it_interrupt	M A
	r_cg_it.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
8-bit interval timer	r_cg_it8bit.c	R_IT8bitm_Channeln_Create R_IT8bitm_Channeln_Start R_IT8bitm_Channeln_Stop R_IT8bitm_Channeln_Set_PowerOff R_IT8bitm_Set_PowerOff	A A A M M
	r_cg_it8bit_user.c	R_IT8bitm_Channeln_Create_UserInit r_it8bitm_channeln_interrupt	M A
	r_cg_it8bit.h	-	-
16-bit wakeup timer	r_cg_timer.c	R_WUTM_Create R_WUTM_Start R_WUTM_Stop R_WUTM_Set_PowerOff	A A A M
	r_cg_timer_user.c	R_WUTM_Create_UserInit r_wutm_interrupt	M A
	r_cg_timer.h	-	-
Clock output/buzzer output controller	r_cg_pclbuz.c	R_PCLBUZn_Create R_PCLBUZn_Start R_PCLBUZn_Stop R_PCLBUZ_Set_PowerOff	A A A M
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit	M
	r_cg_pclbuz.h	-	-
Watchdog timer	r_cg_wdt.c	R_WDT_Create R_WDT_Restart	A A
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_interrupt	M A
	r_cg_wdt.h	-	-
24-bit DS A/D con-verter with program-mable gain instrumentation amplifier	r_cg_pga_dsad.c	R_PGA_DSAD_Create R_PGA_DSAD_Start R_PGA_DSAD_Stop R_PGA_DSAD_Set_PowerOff R_PGA_DSAD_Get_AverageResult R_PGA_DSAD_Get_Result R_PGA_DSAD_CAMP_OffsetTrimming	A A A M A A A
	r_cg_pga_dsad_user.c	R_PGA_DSAD_Create_UserInit r_pga_dsad_interrupt_conversion r_pga_dsad_interrupt_scan r_pga_dsad_conversion_interrupt r_pga_dsad_scan_interrupt	M A A A A
	r_cg_pga_dsad.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
A/D converter	r_cg_adc.c	R_ADC_Create R_ADC_Set_OperationOn R_ADC_Set_OperationOff R_ADC_Start R_ADC_Stop R_ADC_Reset R_ADC_Set_PowerOff R_ADC_Set_ADChannel R_ADC_Set_SnoozeOn R_ADC_Set_SnoozeOff R_ADC_Set_TestChannel R_ADC_Get_Result R_ADC_Get_Result_8bit	A A A A A M M M M M M A A
	r_cg_adc_user.c	R_ADC_Create_UserInit r_adc_interrupt	M A
	r_cg_adc.h	-	-
Configurable amplifier	r_cg_camp.c	R_CAMP_Create R_CAMPn_Start R_CAMPn_Stop R_CAMP_Set_PowerOff	A A A M
	r_cg_camp_user.c	R_CAMP_Create_UserInit	M
	r_cg_camp.h	-	-
Temperature sensor	r_cg_tmps.c	R_TMPS_Create R_TMPS_Start R_TMPS_Stop R_TMPS_Reset R_TMPS_Set_PowerOff	A A A M M
	r_cg_tmps_user.c	R_TMPS_Create_UserInit	M
	r_cg_tmps.h	-	-
24-bit DS A/D converter	r_cg_dsadc.c	R_DSADC_Create R_DSADC_Set_OperationOn R_DSADC_Set_OperationOff R_DSADC_Start R_DSADC_Stop R_DSADC_Reset R_DSADC_Set_PowerOff R_DSADC_Channeln_Get_Result R_DSADC_Channeln_Get_Result_16bit	A A A A A M M A A
	r_cg_dsadc_user.c	R_DSADC_Create_UserInit r_dsadc_interrupt r_dsadzcn_interrupt	M A A
	r_cg_dsadc.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
D/A converter	r_cg_dac.c	R_DAC_Create R_DACn_Start R_DACn_Stop R_DAC_Set_PowerOff R_DACn_Set_ConversionValue R_DAC_Change_OutputVoltage_8bit R_DAC_Change_OutputVoltage R_DACn_Create R_DAC_Reset	A A A M A A A A A
	r_cg_dac_user.c	R_DAC_Create_UserInit R_DACn_Create_UserInit	M M
	r_cg_dac.h	-	-
Programmable gain amplifier	r_cg_pga.c	R_PGA_Create R_PGA_Start R_PGA_Stop R_PGA_Reset R_PGA_Set_PowerOff	A A A M M
	r_cg_pga_user.c	R_PGA_Create_UserInit	M
	r_cg_pga.h	-	-
Comparator	r_cg_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop R_COMP_Reset R_COMP_Set_PowerOff	A A A M M
	r_cg_comp_user.c	R_COMP_Create_UserInit r_compn_interrupt	M A
	r_cg_comp.h	-	-
Comparator/Pro-grammableGain-Amplifier	r_cg_comppga.c	R_COMPPGA_Create R_COMPPGA_Set_PowerOff R_COMPn_Start R_COMPn_Stop R_PGA_Start R_PGA_Stop R_PWMOPT_Start R_PWMOPT_Stop	A M A A A A A A
	r_cg_comppga_user.c	R_COMP_Create_UserInit r_compn_interrupt R_COMPPGA_Create_UserInit r_compn_interrupt	M A M A
	r_cg_comppga.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Serial array unit	r_cg_serial.c or r_cg_sau.c	R_SAUm_Create	A
		R_SAUm_Reset	M
		R_SAUm_Set_PowerOff	M
		R_SAUm_Set_SnoozeOn	M
		R_SAUm_Set_SnoozeOff	M
		R_UARTn_Create	A
		R_UARTn_Start	A
		R_UARTn_Stop	A
		R_UARTn_Send	A
		R_UARTn_Receive	A
		R_CSImn_Create	A
		R_CSImn_Start	A
		R_CSImn_Stop	A
		R_CSImn_Send	A
		R_CSImn_Receive	A
		R_CSImn_Send_Receive	A
		R_IICmn_Create	A
		R_IICmn_StartCondition	A
		R_IICmn_StopCondition	A
		R_IICmn_Stop	A
		R_IICmn_Master_Send	A
		R_IICmn_Master_Receive	A
	r_cg_serial_user.c or r_cg_sau_user.c	R_SAUm_Create_UserInit	M
		r_uartn_interrupt_send	A
		r_uartn_interrupt_receive	A
		r_uartn_interrupt_error	A
		r_uartn_callback_sendend	A
		r_uartn_callback_receiveend	A
		r_uartn_callback_error	A
		r_uartn_callback_softwareoverrun	A
		r_csimn_interrupt	A
		r_csimn_callback_sendend	A
		r_csimn_callback_receiveend	A
		r_csimn_callback_error	A
		r_iicmn_interrupt	A
		r_iicmn_callback_master_sendend	A
		r_iicmn_callback_master_receiveend	A
		r_iicmn_callback_master_error	A
	r_cg_serial.h or r_cg_sau.h	-	-
Serial array unit 4	r_cg_serial.c	R_DALIn_Create	A
		R_DALIn_Start	A
		R_DALIn_Stop	A
		R_DALIn_Send	A
		R_DALIn_Receive	A
	r_cg_serial_user.c	r_dalin_interrupt_send	A
		r_dalin_interrupt_receive	A
		r_dalin_interrupt_error	A
		r_dalin_callback_sendend	A
		r_dalin_callback_receiveend	A
		r_dalin_callback_error	A
		r_dalin_callback_softwareoverrun	A
	r_cg_serial.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Asynchronous serial interface LIN-UART	r_cg_serial.c	R_UARTFn_Create R_UARTFn_Start R_UARTFn_Stop R_UARTFn_Set_PowerOff R_UARTFn_Send R_UARTFn_Receive R_UARTFn_Set_DataComparisonOn R_UARTFn_Set_DataComparisonOff	A A A M A A A A
	r_cg_serial_user.c	R_UARTFn_Create_UserInit r_uartfn_interrupt_send r_uartfn_interrupt_receive r_uartfn_interrupt_error r_uartfn_callback_sendend r_uartfn_callback_receiveend r_uartfn_callback_error r_uartfn_callback_softwareoverrun r_uartfn_callback_expbitdetect r_uartfn_callback_idmatch	M A A A A A A A A A
	r_cg_serial.h	-	-
Serial interface IICA	r_cg_serial.c or r_cg_iica.c	R_IICAn_Create R_IICAn_StopCondition R_IICAn_Stop R_IICAn_Reset R_IICAn_Set_PowerOff R_IICAn_Master_Send R_IICAn_Master_Receive R_IICAn_Slave_Send R_IICAn_Slave_Receive R_IICAn_Set_SnoozeOn R_IICAn_Set_SnoozeOff R_IICAn_Set_WakeupOn R_IICAn_Set_WakeupOff	A M A M M A A A A A A A A
	r_cg_serial_user.c or r_cg_iica_user.c	R_IICAn_Create_UserInit r_iican_interrupt r_iican_callback_master_sendend r_iican_callback_master_receiveend r_iican_callback_master_error r_iican_callback_slave_sendend r_iican_callback_slave_receiveend r_iican_callback_slave_error r_iican_callback_getstopcondition	M A A A A A A A M
	r_cg_serial.h or r_cg_iica.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
LCD controller/ driver	r_cg_lcd.c	R_LCD_Create R_LCD_Start R_LCD_Stop R_LCD_Set_VoltageOn R_LCD_Set_VoltageOff R_LCD_Set_PowerOff R_LCD_VoltageOn R_LCD_VoltageOff	A A A A A A A A
	r_cg_lcd_user.c	R_LCD_Create_UserInit r_lcd_interrupt	M A
	r_cg_lcd.h	-	-
Sound generator	r_cg_sg.c	R_SG_Create R_SG_Start R_SG_Stop	A A A
	r_cg_sg_user.c	R_SG_Create_UserInit r_sg_interrupt	M A
	r_cg_sg.h	-	-
DMA controller	r_cg_dmac.c	R_DMACn_Create R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTriggerOn	A A A A A
	r_cg_dmac_user.c	R_DMACn_Create_UserInit R_DMAC_Create_UserInit r_dmacn_interrupt	M M A
	r_cg_dmac.h	-	-
Data transfer con-troller	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff R_DTCDn_Start R_DTCDn_Stop	A A A M A A
	r_cg_dtc_user.c	R_DTC_Create_UserInit	M
	r_cg_dtc.h	-	-
Event link controller	r_cg_elc.c	R_ELC_Create R_ELC_Stop	A A
	r_cg_elc_user.c	R_ELC_Create_UserInit	M
	r_cg_elc.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Interrupt functions	r_cg_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_INTCLRn_Start R_INTCLRn_Stop R_INTRTCICn_Start R_INTRTCICn_Stop R_INTFO_Start R_INTFO_Stop R_INTFO_ClearFlag	A A A A A A A A A A
	r_cg_intc_user.c	R_INTC_Create_UserInit r_intcn_interrupt r_intclrn_interrupt r_intrtcicn_interrupt r_intfo_interrupt	M A A A A
	r_cg_intc.h	-	-
Key interrupt function	r_cg_intc.c or r_cg_key.c	R_KEY_Create R_KEY_Start R_KEY_Stop	A A A
	r_cg_intc_user.c or r_cg_key_user.c	R_KEY_Create_UserInit r_key_interrupt	M A
	r_cg_intc.h or r_cg_key.h	-	-
Voltage detector	r_cg_lvd.c	R_LVD_Create R_LVD_InterruptMode_Start R_LVD_Start_VDD R_LVD_Start_VBAT R_LVD_Start_VRTC R_LVD_Start_EXLVD R_LVD_Stop_VDD R_LVD_Stop_VBAT R_LVD_Start_VRTC R_LVD_Stop_EXLVD R_LVI_Create R_LVI_InterruptMode_Start	A A A A A A A A A A A A
	r_cg_lvd_user.c	R_LVD_Create_UserInit r_lvd_interrupt r_lvd_vddinterrupt r_lvd_vbatinterrupt r_lvd_vrtcinterrupt r_lvd_exlvdinterrupt R_LVI_Create_UserInit r_lvi_interrupt	M A A A A A M A
	r_cg_lvd.h	-	-
Battery backup function	r_cg_bup.c	R_BUP_Create R_BUP_Start R_BUP_Stop	A A A
	r_cg_bup_user.c	R_BUP_Create_UserInit r_bup_interrupt	M A
	r_cg_bup.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
Oscillation stop detector	r_cg_osdc.c	R_OSDC_Create R_OSDC_Start R_OSDC_Stop R_OSDC_Set_PowerOff R_OSDC_Reset	A A A M M
	r_cg_osdc_user.c	R_OSDC_Create_UserInit r_osdc_interrupt	M A
	r_cg_osdc.h	-	-
SPI interface	r_cg_saic.c or r_cg_spi.c	R_SAIC_Create R_SAIC_Write R_SAIC_Read R_SPI_Create R_SPI_Write R_SPI_Read	A A M A A A
	r_cg_saic_user.c or r_cg_spi_user.c	R_SAIC_Create_UserInit R_SPI_Create_UserInit	M M
	r_cg_saic.h or r_cg_spi.h	-	-
Operational amplifier	r_cg_opamp.c	R_OPAMP_Create R_OPAMP_Set_ReferenceCurcuitOn R_OPAMP_Set_ReferenceCircuitOff R_OPAMPn_Start R_OPAMPn_Stop R_OPAMPn_Set_PrechargeOn R_OPAMPn_Set_PrechargeOff	A A A A A M M
	r_cg_saic_user.c	R_OPAMP_Create_UserInit	M
	r_cg_saic.h	-	-
Data operation circuit	r_cg_doc.c	R_DOC_Create R_DOC_SetMode R_DOC_WriteData R_DOC_GetResult R_DOC_ClearFlag R_DOC_Set_PowerOff R_DOC_Reset	A A A A A M M
	r_cg_doc_user.c	R_DOC_Create_UserInit r_doc_interrupt	M A
	r_cg_doc.h	-	-
32-bit Multiply-accumulator	r_cg_mac32bit.c	R_MAC32Bit_Create R_MAC32Bit_Reset R_MAC32Bit_Set_PowerOff R_MAC32bit_MULUnsigned R_MAC32Bit_MULSigned R_MAC32Bit_MACUnsigned R_MAC32Bit_MACSigned	A M M M M M M
	r_cg_mac32bit_user.c	R_MAC32Bit_Create_UserInit r_mac32bit_interrupt_flow	M A
	r_cg_mac32bit.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
12-bit A/D converter	r_cg_12adc.c	R_12ADC_Create R_12ADC_Start R_12ADC_Stop R_12ADC_Get_ValueResult R_12ADC_Set_ADChannel R_12ADC_TemperatureSensorOutput_On R_12ADC_TemperatureSensorOutput_Off R_12ADC_InternalReferenceVoltage_On R_12ADC_InternalReferenceVoltage_Off R_12ADC_Set_PowerOff	A A A A M M M M M M
	r_cg_12adc_user.c	R_12ADC_Create_UserInit r_12adc_interrupt	M A
	r_cg_12adc.h	-	-
12-bit D/A converter	r_cg_12da.c	R_12DA_Create R_12DAn_Start R_12DAn_Stop R_12DA_Set_PowerOff R_12DAn_Set_ConversionValue	A A A M A
	r_cg_12da_user.c	R_12DA_Create_UserInit	M
	r_cg_12da.h	-	-
Operational amplifier and Analog switch	r_cg_ampansw.c	R_AMPANSW_Create R_OPAMPm_Set_ReferenceCurcuitOn R_OPAMPm_Set_ReferenceCircuitOff R_OPAMPm_Start R_OPAMPm_Stop R_ANSW_ChargePumpm_On R_ANSW_ChargePumpm_Off	A A A A A A A
	r_cg_ampansw_user.c	R_AMPANSW_Create_UserInit	M
	r_cg_ampansw.h	-	-
Voltage reference	r_cg_vr.c	R_VR_Create R_VR_Start R_VR_Stop	A A A
	r_cg_vr_user.c	R_VR_Create_UserInit	M
	r_cg_vr.h	-	-
Sampling output timer detector	r_cg_smotd.c	R_SMOTD_Create R_SMOTD_Start R_SMOTD_Stop R_SMOTD_Set_PowerOff	A A A M
	r_cg_smotd_user.c	R_SMOTD_Create_UserInit r_smotd_counterA_interrupt r_smotd_counterB_interrupt r_smotd_smpn_interrupt	M A A A
	r_cg_smotd.h	-	-

Peripheral Function	File Name	API Function Name	Out-put (*1)
External signal sampler	r_cg_exsd.c	R_EXSD_Create R_EXSD_Start R_EXSD_Stop R_EXSD_Set_PowerOff	A A A M
	r_cg_exsd_user.c	R_EXSD_Create_UserInit r_exsd_interrupt	M A
	r_cg_exsd.h	-	-
Serial interface UARTMG	r_cg_uartmg.c	R_UARTMGn_Create R_UARTMGn_Start R_UARTMGn_Stop R_UARTMGn_Set_PowerOff R_UARTMGn_Send R_UARTMGn_Receive	A A A M A A
	r_cg_uartmg_user.c	R_UARTMGn_Create_UserInit r_uartmgn_interrupt_send r_uartmgn_interrupt_receive r_uartmgn_interrupt_error r_uartmgn_callback_sendend r_uartn_callback_receiveend r_uartmgn_callback_error r_uartmgn_callback_softwareoverrun	M A A A A A A A
	r_cg_uartmg.h	-	-
Amplifier unit	r_cg_amp.c	R_AMP_Create R_AMP_Set_PowerOn R_AMP_Set_PowerOff R_PGA1_Start R_OPAMPn_Stop R_AMPn_Start R_AMPn_Stop	A A M A A A A
	r_cg_amp_user.c	R_AMP_Create_UserInit	M
	r_cg_amp.h	-	-
Data flash libraries	r_cg_pfdl.c	R_FDL_Create R_FDL_Open R_FDL_Close R_FDL_Write R_FDL_Read R_FDL_Erase	A A A A A A
	r_cg_pfdl.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

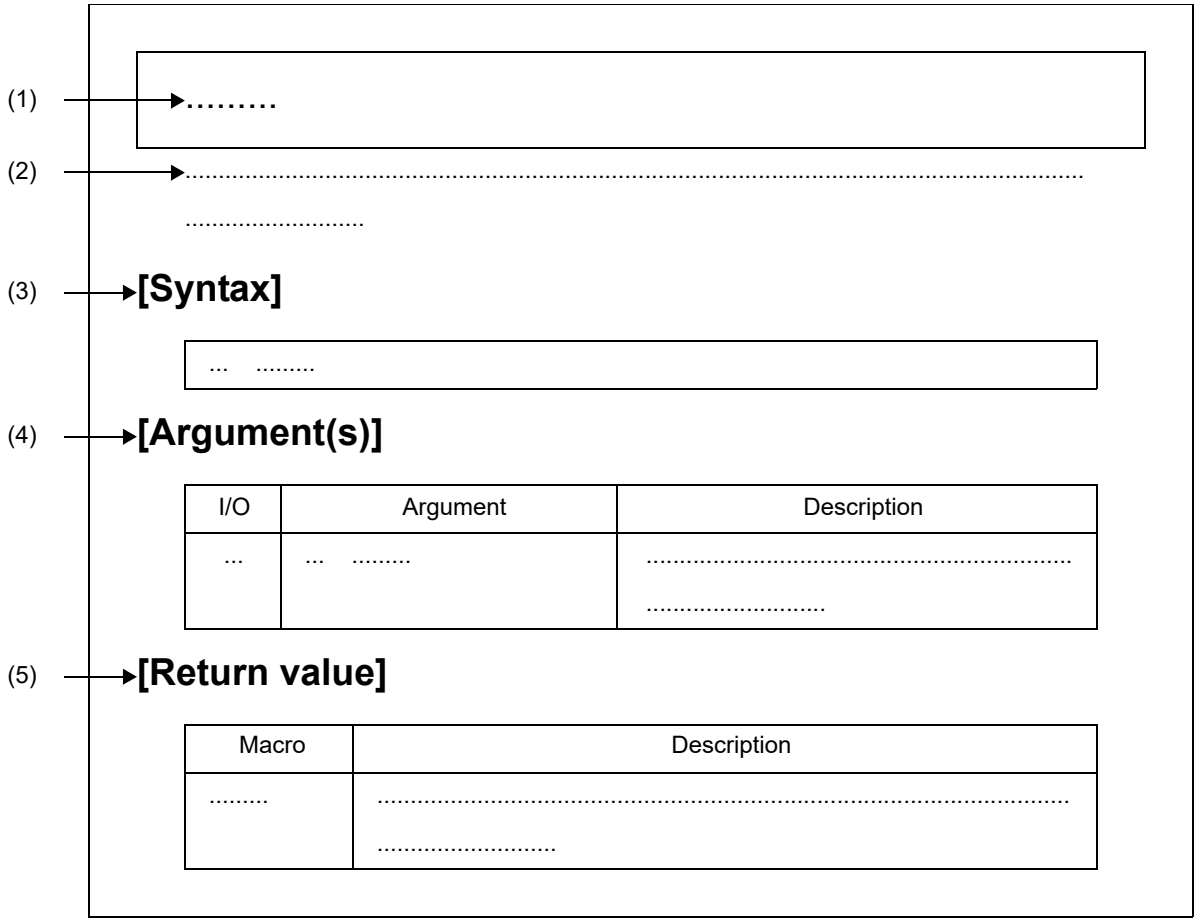
Below are the naming conventions for API functions output by the Code Generator.

- 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.

Table 3.1 API Functions: [Common]

API Function Name	Function
hdwinit	Performs initialization necessary to control the various hardwares. This API is automatically called from the startup routine of Renesas-made compiler.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.

hdwinit

Performs initialization necessary to control the various hardwares.

Remark Call this API function from the startup routine.

[Syntax]

```
void    hdwinit ( 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 hdwinit callback routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark Call this API function from the startup 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.

3.2.2 Clock generator

Below is a list of API functions output by the Code Generator for clock generator (include reset function, on-chip debug function, etc.) use.

Table 3.2 API Functions: [Clock Generator]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).
r_cgc_ram_ecc_interrupt	Performs processing in response to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.
r_cgc_stackpointer_interrupt	Performs processing in response to the stackpointer overflow/underflow interrupt INTSPM.
r_cgc_clockmonitor_interrupt	Performs processing in response to the clock monitor interrupt INTCLM.
R_CGC_Get_ResetSource	Performs processing in response to RESET signal.
R_CGC_Set_ClockMode	Changes the CPU clock/peripheral hardware clock.
R_CGC_RAMECC_Start	Starts the RAM-ECC function.
R_CGC_RAMECC_Stop	Ends the RAM-ECC function.
R_CGC_StackPointer_Start	Starts the CPU stack pointer monitor function.
R_CGC_StackPointer_Stop	Ends the CPU stack pointer monitor function.
R_CGC_ClockMonitor_Start	Starts the clock monitor.
R_CGC_ClockMonitor_Stop	Ends the clock monitor.

R_CGC_Create

Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).

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_ram_ecc_interrupt

Performs processing in response to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.

Remark This API function is called as the interrupt process corresponding to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_cgc_ram_ecc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_cgc_ram_ecc_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_cgc_stackpointer_interrupt

Performs processing in response to the stack pointer overflow/underflow interrupt INTSPM.

Remark This API function is called as the interrupt process corresponding to the stack pointer overflow/underflow interrupt INTSPM.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_cgc_stackpointer_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_cgc_stackpointer_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_clockmonitor_interrupt

Performs processing in response to the clock monitor interrupt INTCLM.

Remark This API function is called as the interrupt process corresponding to the clock monitor interrupt INTCLM.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_cgc_clockmonitor_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_cgc_clockmonitor_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Get_ResetSource

Performs processing in response to RESET signal.

[Syntax]

```
void    R_CGC_Get_ResetSource ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Changes the CPU clock/peripheral hardware clock.

[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 generator type HIOCLK: High-speed onchip oscillator SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBXT1CLK: XT1 clock SUBEXTCLK: External subsystem clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ERROR2	Exit with error (abend)
MD_ERROR3	Exit with error (abend)
MD_ERROR4	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

R_CGC_RAMECC_Start

Starts the RAM-ECC function.

[Syntax]

```
void    R_CGC_RAMECC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_RAMECC_Stop

Ends the RAM-ECC function.

[Syntax]

```
void    R_CGC_RAMECC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_StackPointer_Start

Starts the CPU stack pointer function.

[Syntax]

```
void    R_CGC_StackPointer_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_StackPointer_Stop

Ends the CPU stack pointer function.

[Syntax]

```
void    R_CGC_StackPointer_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_ClockMonitor_Start

Starts the clock monitor.

[Syntax]

```
void    R_CGC_ClockMonitor_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_ClockMonitor_Stop

Ends the clock monitor.

[Syntax]

```
void    R_CGC_ClockMonitor_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.3 Port functions

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

Table 3.3 API Functions: [Port Functions]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control the port functions.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the port functions.

R_PORT_Create

Performs initialization necessary to control the port functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the port functions.

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.

3.2.4 High-speed on-chip Oscillator clock Frequency Correction function

Below is a list of API functions output by the Code Generator for the High-speed on-chip Oscillator clock Frequency Correction function use.

Table 3.4 API Functions: [High-speed on-chip Oscillator clock Frequency Correction function]

API Function Name	Function
R_HOFC_Create	Performs initialization necessary to control the High-speed on-chip Oscillator clock Frequency Correction function.
R_HOFC_Create_UserInit	Performs user-defined initialization relating to the High-speed on-chip Oscillator clock Frequency Correction function.
r_hofc_interrupt	Performs processing in response to the timer interrupt.
R_HOFC_Start	Starts the count for the High-speed on-chip Oscillator clock Frequency Correction function.
R_HOFC_Stop	Ends the count for the High-speed on-chip Oscillator clock Frequency Correction function.

R_HOFC_Create

Performs initialization necessary to control the High-speed on-chip Oscillator clock Frequency Correction function.

[Syntax]

```
void    R_HOFC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Create_UserInit

Performs user-defined initialization relating to the High-speed on-chip Oscillator clock Frequency Correction function.

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

[Syntax]

```
void    R_HOFC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_hofc_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the High-speed on-chip Oscillator clock Frequency Correction function interrupt INTCR.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_hofc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_hofc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Start

Starts the count for the High-speed on-chip Oscillator clock Frequency Correction function.

[Syntax]

```
void    R_HOFC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Stop

Ends the count for the High-speed on-chip Oscillator clock Frequency Correction function.

[Syntax]

```
void    R_HOFC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.5 Timer array unit

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

Table 3.5 API Functions: [Timer Array Unit]

API Function Name	Function
R_TAUm_Create	Performs initialization necessary to control the timer array unit.
R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.
r_taum_channeln_interrupt	Performs processing in response to the timer interrupt INTTM*mn*.
r_taum_channeln_higher8bits_interrupt	Performs processing in response to the timer interrupt INTTM*mn*H.
R_TAUm_Channeln_Start	Starts the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel *n*.
R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel *n*.
R_TAUm_Channeln_Stop	Ends the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel *n*.
R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel *n*.
R_TAUm_Reset	Reset the timer array unit.
R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
R_TAUm_Channeln_Set_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.

R_TAU*m*_Create

Performs initialization necessary to control the timer array unit.

[Syntax]

```
void    R_TAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Create_UserInit

Performs user-defined initialization relating to the timer array unit.

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

[Syntax]

```
void    R_TAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_tau*m*_channel*n*_interrupt

Performs processing in response to the timer interrupt INTTM*mn*.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM*mn*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tau*m*_channel*n*_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tau*m*_channel*n*_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tau*m*_channel*n*_higher8bits_interrupt

Performs processing in response to the timer interrupt INTTM*mn*H.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM*mn*H.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_taum_channeln_higher8bits_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_taum_channeln_higher8bits_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Start

Starts the count for channel *n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or external event counter).

[Syntax]

```
void    R_TAUm_Channeln_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Start

Starts the count (higher 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Syntax]

```
void    R_TAUm_Channeln_Higher8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Start

Starts the count (lower 8-bit) for channel *n*.

Remarks 1. This API function can only be called when the timer array unit is used as a 8-bit timer.

Remarks 2. The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, external event counter, or delay counter).

[Syntax]

```
void    R_TAUm_Channeln_Lower8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Stop

Ends the count for channel *n*.

[Syntax]

```
void    R_TAUm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Stop

Ends the count (higher 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Syntax]

```
void    R_TAUm_Channeln_Higher8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Stop

Ends the count (lower 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Syntax]

```
void    R_TAUm_Channeln_Lower8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Reset

Reset the timer array unit.

[Syntax]

```
void    R_TAUm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Set_PowerOff

Halts the clock supplied to the timer array unit.

Remark Calling this API function changes the timer array unit to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TAUm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Get_PulseWidth

Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_TAUm_Channeln_Get_PulseWidth ( uint32_t * const width );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const width;`	Pointer to an area to store the measurement width (0x0 to 0x1FFFF)

[Return value]

None.

R_TAU*m*_Channel*n*_Set_SoftwareTriggerOn

Generates the trigger (software trigger) for one-shot pulse output.

[Syntax]

```
void    R_TAUm_Channeln_Set_SoftwareTriggerOn ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.6 Timer RJ

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

Table 3.6 API Functions: [Timer RJ]

API Function Name	Function
R_TMR_RJn_Create	Performs initialization necessary to control the 16-bit timer RJ0.
R_TMR_RJn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ0.
r_tmr_rjn_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RJn_Start	Starts the count for 16-bit timer RJ0.
R_TMR_RJn_Stop	Ends the count for 16-bit timer RJ0.
R_TMR_RJn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ0.
R_TMR_RJn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ0.
R_TMRJn_Create	Performs initialization necessary to control the 16-bit timer RJ0.
R_TMRJn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ0.
r_tmrjn_interrupt	Performs processing in response to the timer interrupt.
R_TMRJn_Start	Starts the count for 16-bit timer RJ0.
R_TMRJn_Stop	Ends the count for 16-bit timer RJ0.
R_TMRJn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ0.
R_TMRJn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ0.

R_TMR_RJ*n*_Create

Performs initialization necessary to control the 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMR_RJn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ*n*.

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

[Syntax]

```
void    R_TMR_RJn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_rj*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmr_rjn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmr_rjn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Start

Starts the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMR_RJn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Stop

Ends the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMR_RJn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ*n*.

Remark Calling this API function changes the 16-bit timer RJ*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RJn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ*n*.

Remarks 1. This API function can only be called when the 16-bit timer RJ*n* is being used for pulse width measurement mode / pulse period measurement mode.

Remarks 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_TMR_RJn_Get_PulseWidth ( uint32_t * const active_width );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const active_width;`	Pointer to an area storing the active level width that was read from the TRJ*n*IO pin

[Return value]

None.

R_TMRJ*n*_Create

Performs initialization necessary to control the 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMRJn_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ*n*.

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

[Syntax]

```
void    R_TMRJn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmrj*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmrjn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmrjn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Start

Starts the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMRJn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Stop

Ends the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMRJn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ*n*.

Remark Calling this API function changes the 16-bit timer RJ*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRJn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ*n*.

Remarks 1. This API function can only be called when the 16-bit timer RJ*n* is being used for pulse width measurement mode / pulse period measurement mode.

Remarks 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_TMRJn_Get_PulseWidth ( uint32_t * const active_width );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const active_width;`	Pointer to an area storing the active level width that was read from the TRJ*n*IO pin

[Return value]

None.

3.2.7 Timer RD

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

Table 3.7 API Functions: [Timer RD]

API Function Name	Function
R_TMR_RDn_Create	Performs initialization necessary to control the 16-bit timer RD*n*.
R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD*n*.
r_tmr_rdn_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RDn_Start	Starts the count for 16-bit timer RD*n*.
R_TMR_RDn_Stop	Ends the count for 16-bit timer RD*n*.
R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD*n*.
R_TMR_RDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD*n*,
R_TMR_RDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD*n*.
R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD*n*.
R_TMRDn_Create	Performs initialization necessary to control the 16-bit timer RD*n*.
R_TMRDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD*n*.
r_tmrdn_interrupt	Performs processing in response to the timer interrupt.
R_TMRDn_Start	Starts the count for 16-bit timer RD*n*.
R_TMRDn_Stop	Ends the count for 16-bit timer RD*n*.
R_TMRDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD*n*.
R_TMRDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD*n*,
R_TMRDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD*n*.
R_TMRDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD*n*.
R_TMRD_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD.

R_TMR_RD*n*_Create

Performs initialization necessary to control the 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RD*n*.

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

[Syntax]

```
void    R_TMR_RDn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_rd*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmr_rdn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmr_rdn_interrupt ( void );
```

Remark [*n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Start

Starts the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Stop

Ends the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD*n*.

Remark Calling this API function changes the 16-bit timer RD*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RDn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_ForcedOutput_Start

Starts the pulse output forced cutoff for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_ForcedOutput_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_ForcedOutput_Stop

Ends the pulse output forced cutoff for 16-bit timer RD*n*.

Remark This API function can only be called when the 16-bit timer RD*n* is the count to stopped (the TSTART bit in the timer RD start register (TRDSTR) is 0).

[Syntax]

```
void    R_TMR_RDn_ForcedOutput_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RD*n*.

Remarks 1. This API function can only be called when the 16-bit timer RD*n* is being used for input capture function.

Remarks 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_timer.h"
MD_STATUS   R_TMR_RDn_Get_PulseWidth ( uint32_t * const active_width, uint32_t * const
inactive_width, timer_channel_t channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const active_width;`	Pointer to an area storing the active level width that was read
O	`uint32_t * const inactive_width;`	Pointer to an area storing the inactive level width that was read
I	`timer_channel_t channel;`	Pin to read TMCHANNELA: TRDIOA*n* pin TMCHANNELB: TRDIOB*n* pin TMCHANNELC: TRDIOC*n* pin TMCHANNELD: TRDIOD*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMRD*n*_Create

Performs initialization necessary to control the 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RD*n*.

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

[Syntax]

```
void    R_TMRDn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmrd*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmrdn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmrdn_interrupt ( void );
```

Remark [*n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Start

Starts the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Stop

Ends the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD*n*.

Remark Calling this API function changes the 16-bit timer RD*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRDn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_ForcedOutput_Start

Starts the pulse output forced cutoff for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_ForcedOutput_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_ForcedOutput_Stop

Ends the pulse output forced cutoff for 16-bit timer RD*n*.

Remark This API function can only be called when the 16-bit timer RD*n* is the count to stopped (the TSTART bit in the timer RD start register (TRDSTR) is 0).

[Syntax]

```
void    R_TMRDn_ForcedOutput_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RD*n*.

Remarks 1. This API function can only be called when the 16-bit timer RD*n* is being used for input capture function.

Remarks 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_timer.h"
MD_STATUS   R_TMRDn_Get_PulseWidth ( uint32_t * const active_width, uint32_t * const
inactive_width, timer_channel_t channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const active_width;`	Pointer to an area storing the active level width that was read
O	`uint32_t * const inactive_width;`	Pointer to an area storing the inactive level width that was read
I	`timer_channel_t channel;`	Pin to read TMCHANNELA: TRDIOA*n* pin TMCHANNELB: TRDIOB*n* pin TMCHANNELC: TRDIOC*n* pin TMCHANNELD: TRDIOD*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMRD_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD.

Remark Calling this API function changes the 16-bit timer RD*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.8 Timer RG

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

Table 3.8 API Functions: [Timer RG]

API Function Name	Function
R_TMR_RG0_Create	Performs initialization necessary to control the 16-bit timer RG0.
R_TMR_RG0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RG0.
r_tmr_rg0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RG0_Start	Starts the count for 16-bit timer RG0.
R_TMR_RG0_Stop	Ends the count for 16-bit timer RG0.
R_TMR_RG0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RG0.
R_TMR_RG0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RG0.

R_TMR_RG0_Create

Performs initialization necessary to control the 16-bit timer RG0.

[Syntax]

```
void    R_TMR_RG0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RG0.

Remark This API function is called as the R_TMR_RG0_Create callback routine.

[Syntax]

```
void    R_TMR_RG0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_rg0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmr_rg0_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmr_rg0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Start

Starts the count for 16-bit timer RG0.

[Syntax]

```
void    R_TMR_RG0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Stop

Ends the count for 16-bit timer RG0.

[Syntax]

```
void    R_TMR_RG0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Set_PowerOff

Halts the clock supplied to the 16-bit timer RG0.

Remark Calling this API function changes the 16-bit timer RG0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RG0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Get_PulseWidth

Reads the pulse width of the 16-bit timer RG0.

Remarks 1. This API function can only be called when the 16-bit timer RG0 is being used for input capture function.

Remarks 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_timer.h"
MD_STATUS   R_TMR_RG0_Get_PulseWidth ( uint32_t * const active_width, uint32_t * const
inactive_width, timer_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const active_width;`	Pointer to an area storing the active level width that was read from the TRGIOA pin
O	`uint32_t * const inactive_width;`	Pointer to an area storing the inactive level width that was read from the TRGIOA pin
I	`timer_channel_t channel;`	Pin to read TMCHANNELA: TRGIOA0 pin TMCHANNELB: TRGIOB0 pin

[Return value]

Macro	Description
MD_OK	Normal completion

3.2.9 Timer RX

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

Table 3.9 API Functions: [Timer RX]

API Function Name	Function
R_TMRX_Create	Performs initialization necessary to control the 16-bit timer RX.
R_TMRX_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RX.
r_tmrx_interrupt	Performs processing in response to the timer interrupt.
R_TMRX_Start	Starts the count for 16-bit timer RX.
R_TMRX_Stop	Ends the count for 16-bit timer RX.
R_TMRX_Set_PowerOff	Halts the clock supplied to the 16-bit timer RX.
R_TMRX_Get_BufferValue	Reads the buffer value of TRX resister(16-bit timer RX).

R_TMRX_Create

Performs initialization necessary to control the 16-bit timer RX.

[Syntax]

```
void    R_TMRX_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Create_UserInit

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

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

[Syntax]

```
void    R_TMRX_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmrx_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmrx_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmrx_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Start

Starts the count for 16-bit timer RX.

[Syntax]

```
void    R_TMRX_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Stop

Ends the count for 16-bit timer RX.

[Syntax]

```
void    R_TMRX_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Set_PowerOff

Halts the clock supplied to the 16-bit timer RX.

Remark Calling this API function changes the 16-bit timer RX to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRX_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Get_BufferValue

Reads the buffer value of TRX register (16-bit timer RX).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void   R_TMRX_Get_BufferValue ( uint32_t * const value );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const value;`	Pointer to an area storing the buffer resister value of TRX resister

[Return value]

なし

3.2.10 16-bit timer KB

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

Table 3.10 API Functions: [16-bit Timers KB]

API Function Name	Function
R_TMR_KB_Create	Performs initialization necessary to control the 16-bit timer KB.
R_TMR_KBm_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KB.
r_tmr_kbm_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KBm_Start	Starts the count for 16-bit timer KB.
R_TMR_KBm_Stop	Ends the count for 16-bit timer KB.
R_TMR_KBm_Set_PowerOff	Halts the clock supplied to the 16-bit timer KB.
R_TMR_KBmn_ForcedOutput_Start	Enables input of the trigger signal used for the forced output stop function.
R_TMR_KBmn_ForcedOutput_Stop	Disables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_BatchOverwriteRequestOn	Enables batch overwriting of the compare register.
R_TMR_KBm_ForcedOutput_mn_Start	Enables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_ForcedOutput_mn_Stop	Disables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_Reset	Reset the 16-bit timer KB.

R_TMR_KB_Create

Performs initialization necessary to control the 16-bit timers KB.

[Syntax]

```
void    R_TMR_KB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Create_UserInit

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

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

[Syntax]

```
void    R_TMR_KBm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_kb*m*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmr_kbm_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmr_kbm_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Start

Starts the count for 16-bit timer KB.

[Syntax]

```
void    R_TMR_KBm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Stop

Ends the count for 16-bit timer KB.

[Syntax]

```
void    R_TMR_KBm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Set_PowerOff

Halts the clock supplied to the 16-bit timer KB.

Remark Calling this API function changes the 16-bit timer KB to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_KBm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*mn*_ForcedOutput_Start

Enables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBmn_ForcedOutput_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*mn*_ForcedOutput_Stop

Disables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBmn_ForcedOutput_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_BatchOverwriteRequestOn

Enables batch overwriting of the compare register.

Remark The timing for batch-overwriting the content of the compare register is when a count value and a value set in the compare register are matched or an external trigger is generated after calling this API function.

[Syntax]

```
void    R_TMR_KBm_BatchOverwriteRequestOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_ForcedOutput_*mn*_Start

Enables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBm_ForcedOutput_mn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_ForcedOutput_*mn*_Stop

Disables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBm_ForcedOutput_mn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Reset

Reset the 16-bit timers KB.

[Syntax]

```
void    R_TMR_KBm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.11 16-bit timer KC0

Below is a list of API functions output by the Code Generator for 16-bit timer KC0 use.

Table 3.11 API Functions: [16-bit Timer KC0]

API Function Name	Function
R_TMR_KC0_Create	Performs initialization necessary to control the 16-bit timer KC0.
R_TMR_KC0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KC0.
r_tmr_kc0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KC0_Start	Starts the count for 16-bit timer KC0.
R_TMR_KC0_Stop	Ends the count for 16-bit timer KC0.
R_TMR_KC0_Set_PowerOff	Halts the clock supplied to the 16-bit timer KC0.

R_TMR_KC0_Create

Performs initialization necessary to control the the 16-bit timer KC0.

[Syntax]

```
void    R_TMR_KC0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KC0.

Remark This API function is called as the R_TMR_KC0_Create callback routine.

[Syntax]

```
void    R_TMR_KC0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_kc0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_tmr_kc0_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_tmr_kc0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Start

Starts the count for 16-bit timer KC0.

[Syntax]

```
void    R_TMR_KC0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Stop

Ends the count for 16-bit timer KC0.

[Syntax]

```
void    R_TMR_KC0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Set_PowerOff

Halts the clock supplied to the 16-bit timer KC0.

Remark Calling this API function changes the 16-bit timer KC0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_KC0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.12 16-bit timer KB2

Below is a list of API functions output by the Code Generator for 16-bit timer KB2 use.

Table 3.12 API Functions: [16-bit Timer KB2]

API Function Name	Function
R_KB2m_Create	Performs initialization necessary to control the 16-bit timer KB2.
R_KB2m_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KB2.
r_kb2m_interrupt	Performs processing in response to the timer interrupt INTTKB2*m*.
R_KB2m_Start	Starts the count for 16-bit timer KB2.
R_KB2m_Stop	Ends the count for 16-bit timer KB2.
R_KB2m_Set_PowerOff	Halts the clock supplied to the 16-bit timer KB2.
R_KB2m_Simultaneous_Start	Starts the simultaneous start/stop mode.
R_KB2m_Simultaneous_Stop	Ends the simultaneous start/stop mode.
R_KB2m_Synchronous_Start	Starts the timer start/clear mode.
R_KB2m_Synchronous_Stop	Ends the timer start/clear mode.
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	Starts forced output stop function 1 for timer output TKBO*n*0.
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	Ends forced output stop function 1 for timer output TKBO*n*0.
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	Starts forced output stop function 2 for timer output TKBO*n*1.
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	Starts forced output stop function 2 for timer output TKBO*n*1.
R_KB2m_TKBOn0_DitheringFunction_Start	Starts dithering function for timer output TKBO*n*0.
R_KB2m_TKBOn0_DitheringFunction_Stop	Ends dithering function for timer output TKBO*n*0.
R_KB2m_TKBOn1_DitheringFunction_Start	Starts dithering function for timer output TKBO*n*1.
R_KB2m_TKBOn1_DitheringFunction_Stop	Ends dithering function for timer output TKBO*n*1.
R_KB2m_TKBOn0_SmoothStartFunction_Start	Starts smooth start function for timer output TKBO*n*0.
R_KB2m_TKBOn0_SmoothStartFunction_Stop	Ends smooth start function for timer output TKBO*n*0.
R_KB2m_TKBOn1_SmoothStartFunction_Start	Starts smooth start function for timer output TKBO*n*1.
R_KB2m_TKBOn1_SmoothStartFunction_Stop	Ends smooth start function for timer output TKBO*n*1.
R_KB2m_Set_BatchOverwriteRequestOn	Enables batch overwriting of the compare register.

R_KB2*m*_Create

Performs initialization necessary to control the 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KB2.

Remark This API function is called as the R_KB2m_Create callback routine.

[Syntax]

```
void    R_KB2m_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_kb2*m*_interrupt

Performs processing in response to the timer interrupt INTTKB2*m*.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTKB2*m*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_kb2m_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_kb2m_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Start

Starts the count for 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Stop

Ends the count for 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Set_PowerOff

Halts the clock supplied to the 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Simultaneous_Start

Starts the simultaneous start/stop mode.

[Syntax]

```
void    R_KB2m_Simultaneous_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Simultaneous_Stop

Ends the simultaneous start/stop mode.

[Syntax]

```
void    R_KB2m_Simultaneous_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Synchronous_Start

Starts the timer start/clear mode.

[Syntax]

```
void    R_KB2m_Synchronous_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Synchronous_Stop

Ends the timer start/clear mode.

[Syntax]

```
void    R_KB2m_Synchronous_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Start

Starts forced output stop function 1 for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Stop

Ends forced output stop function 1 for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Start

Starts forced output stop function 2 for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Stop

Ends forced output stop function 2 for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_DitheringFunction_Start

Starts dithering function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_DitheringFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_DitheringFunction_Stop

Ends dithering function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_DitheringFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_DitheringFunction_Start

Starts dithering function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_DitheringFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_DitheringFunction_Stop

Ends dithering function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_DitheringFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Start

Starts smooth start function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Stop

Ends smooth start function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Start

Starts smooth start function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Stop

Ends smooth start function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Set_BatchOverwriteRequestOn

Enables batch overwriting of the compare register.

Remark The timing for batch-overwriting the content of the compare register is when a count value and a value set in the compare register are matched or an external trigger is generated after calling this API function.

[Syntax]

```
void    R_KB2m_Set_BatchOverwriteRequestOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.13 Real-time clock

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

Table 3.13 API Functions: [Real-time Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control the real-time clock.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the real-time clock.
r_rtc_interrupt	Performs processing in response to the real-time clock interrupt INTRTC.
R_RTC_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Set_PowerOff	Halts the clock supplied to the real-time clock.
R_RTC_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock.
R_RTC_Set_CounterValue	Sets the counter value of the real-time clock.
R_RTC_Set_CalendarCounterValue	Sets the counter value of the real-time clock.(in the case of the calendar mode setting)
R_RTC_Set_BinaryCounterValue	Sets the counter value of the real-time clock.(in the case of the binary mode setting)
R_RTC_Get_CounterValue	Reads the counter value of the real-time clock.
R_RTC_Get_CalendarCounterValue	Reads the counter value of the real-time clock.(in the case of the calendar mode setting)
R_RTC_Get_BinaryCounterValue	Reads the counter value of the real-time clock.(in the case of the binary mode setting)
R_RTC_Set_ConstPeriodInterruptOn	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
R_RTC_Set_ConstPeriodInterruptOff	Ends the cyclic interrupt function.
r_rtc_callback_constperiod	Performs processing in response to the cyclic interrupt INTRTC.
R_RTC_Set_AlarmOn	Starts the alarm interrupt function.
R_RTC_Set_CalendarAlarmOn	Starts the alarm interrupt function.(in the case of the calendar mode setting)
R_RTC_Set_BinaryAlarmOn	Starts the alarm interrupt function.(in the case of the binary mode setting)
R_RTC_Set_AlarmOff	Ends the alarm interrupt function.
R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
R_RTC_Set_CalendarAlarmValue	Sets the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)
R_RTC_Set_BinaryAlarmValue	Sets the alarm conditions.(in the case of the binary mode setting)
R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).
R_RTC_Get_CalendarAlarmValue	Reads the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)

API Function Name	Function
R_RTC_Get_BinaryAlarmValue	Reads the alarm conditions.(in the case of the binary mode setting)
r_rtc_callback_alarm	Performs processing in response to the alarm interrupt INTRTC.
R_RTC_Set_RTC1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTC1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTCOUTOn	Enables output of the RTCOUT.
R_RTC_Set_RTCOUTOff	Disables output of the RTCOUT.
r_rtc_alarminterrupt	Performs processing in response to the alarm interrupt INTRTCALM.
r_rtc_periodicinterrupt	Performs processing in response to the periodic interrupt INTRTCPRD.
r_rtc_callback_periodic	Performs processing in response to the cyclic interrupt INTRTC.

R_RTC_Create

Performs initialization necessary to control the real-time clock.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the real-time 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_interrupt

Performs processing in response to the real-time clock interrupt INTRTC.

Remark This API function is called as the interrupt process corresponding to the real-time clock interrupt INTRTC.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_rtc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_rtc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_PowerOff

Halts the clock supplied to the real-time clock.

Remarks 1. Calling this API function changes the real-time clock to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remarks 2. This API function stops the clock supply to the real-time clock, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. interval timer).

[Syntax]

```
void    R_RTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_HourSystem

Sets the clock type (12-hour or 24-hour clock) of the real-time clock.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_HourSystem ( rtc_hour_system_t hour_system );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_hour_system_t hour_system;`	Clock type HOUR12: 12-hour clock HOUR24: 24-hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CounterValue

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_CounterValue ( rtc_counter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_counter_value_t counter_write_val;`	Counter value

Remark Below is an example of the structure rtc_counter_value_t (counter value) for the real-time clock.

```
typedef struct {
    uint8_t sec;    /* Second */
    uint8_t min;    /* Minute */
    uint8_t hour;   /* Hour */
    uint8_t day;    /* Day */
    uint8_t week;   /* Weekday (0: Sunday, 6: Saturday) */
    uint8_t month;  /* Month */
    uint16_t year;  /* Year */
} rtc_counter_value_t;
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CalendarCounterValue

Sets the counter value of the real-time clock.(in the case of the calendar mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_CalendarCounterValue ( rtc_counter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_counter_value_t counter_write_val;`	Counter value

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)

Remark If MD_BUSY1 is returned, it may be because the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_BinaryCounterValue

Sets the counter value of the real-time clock.(in the case of the binary mode setting)

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`uint32_t counter_write_val;`	Counter value

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)

Remark If MD_BUSY1 is returned, it may be because the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CounterValue

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Get_CounterValue ( rtc_counter_value_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	`rtc_counter_value_t * const counter_read_val;`	Pointer to structure in which to store the counter value being read

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CalendarCounterValue

Reads the counter value of the real-time clock.(in the case of the calendar mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Get_CalendarCounterValue ( rtc_counter_value_t * const
counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	`rtc_counter_value_t * const counter_read_val;`	Pointer to structure in which to store the counter value being read

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Read failure

R_RTC_Get_BinaryCounterValue

Reads the counter value of the real-time clock.(in the case of the binary mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   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 structure in which to store the counter value being read

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Read failure

R_RTC_Set_ConstPeriodInterruptOn

Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`rtc_int_period_t period;`	Interrupt INTRTC cycle HALFSEC: 0.5 seconds ONESEC: 1 second ONEMIN: 1 minute ONEHOUR: 1 hour ONEDAY: 1 day ONEMONTH: 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_RTC_Set_ConstPeriodInterruptOff

Ends the cyclic interrupt function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_rtc_callback_constperiod

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the cyclic interrupt INTRTC.

[Syntax]

```
static  void    r_rtc_callback_constperiod ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOn

Starts the alarm interrupt function.

[Syntax]

```
void    R_RTC_Set_AlarmOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CalendarAlarmOn

Starts the alarm interrupt function.(in the case of the calendar mode setting)

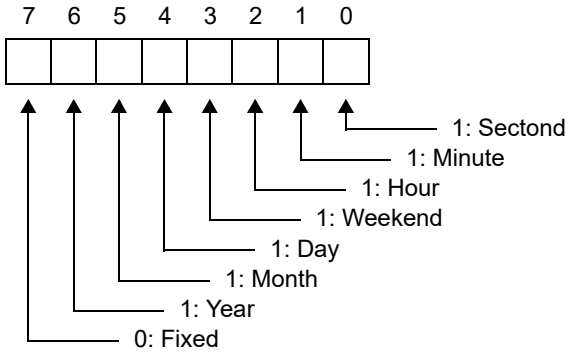
[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarAlarmOn ( uint8_t enb_set );
```

[Argument(s)]

I/O	Argument	Description
I	`uint8_t enb_set;`	Alarm enable

Remark Below is shown the structure enb_set.



[Return value]

None.

R_RTC_Set_BinaryAlarmOn

Starts the alarm interrupt function.(in the case of the binary mode setting)

[Syntax]

```
void    R_RTC_Set_BinaryAlarmOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOff

Ends the alarm interrupt function.

[Syntax]

```
void    R_RTC_Set_AlarmOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmValue

Sets the alarm conditions (weekday, hour, minute).

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Set_AlarmValue ( rtc_alarm_value_t alarm_val );
```

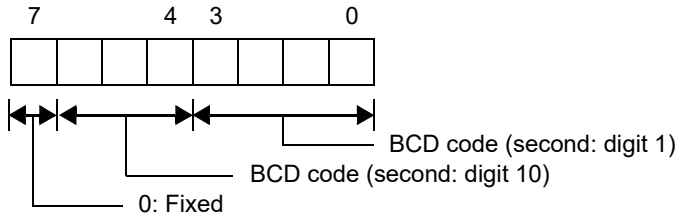
[Argument(s)]

I/O	Argument	Description
I	`rtc_alarm_value_t alarm_val;`	Alarm conditions (weekday, hour, minute)

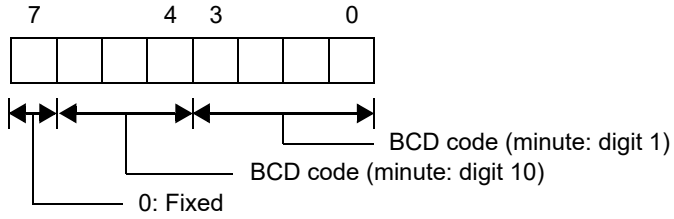
Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).(The structure is different according to the device.)

```
typedef struct {
    uint8_t sec;    /* Second */
    uint8_t min;    /* Minute */
    uint8_t hour;   /* Hour */
    uint8_t week;   /* Weekday (0: Sunday, 6: Saturday) */
    uint8_t day;    /* Day */
    uint8_t month;  /* Month */
    uint16_t year;  /* Year */
} rtc_alarm_value_t;
```

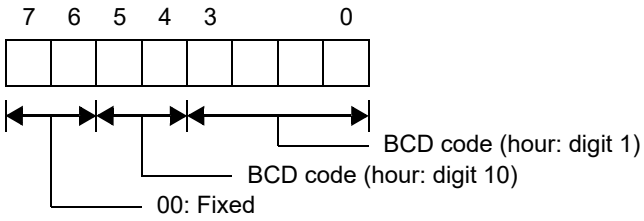
- alarmws (Second)
  Below are shown the meanings of each bit of the structure member alarmws.



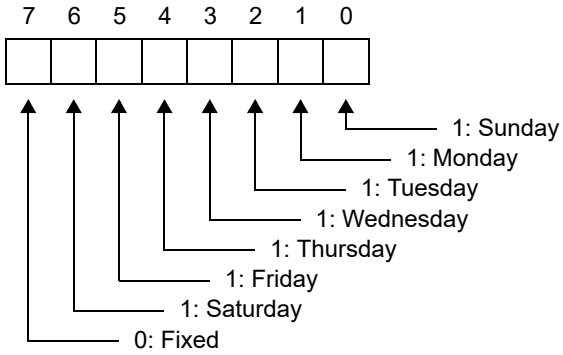
- alarmwm (Minute)
  Below are shown the meanings of each bit of the structure member alarmwm.



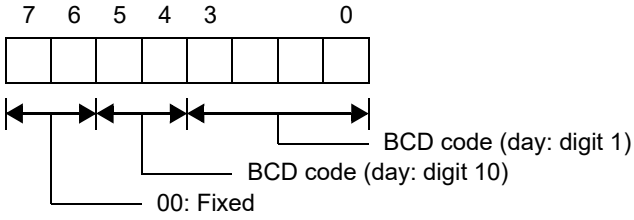
- alarmwh (Hour)
  Below are shown the meanings of each bit of the structure member alarmwh.
  If the real-time clock is set to the 12-hour clock, then bit 5 has the following meaning.
  0: AM
  1: PM



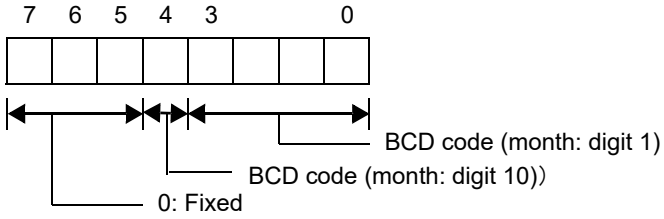
- alarmww (Weekday)
  Below are shown the meanings of each bit of the structure member alarmww.



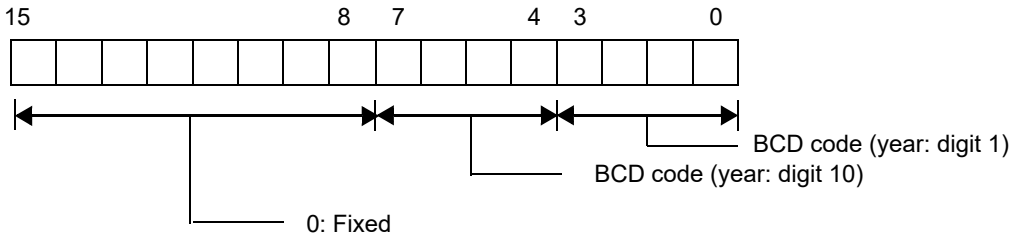
- alarmwd (Day)
  Below are shown the meanings of each bit of the structure member alarmwd.



- alarmwmn (Month)
  Below are shown the meanings of each bit of the structure member alarmwmn.



- alarmwy (Year)
  Below are shown the meanings of each bit of the structure member alarmwmn.



[Return value]

None.

R_RTC_Set_CalendarAlarmValue

Sets the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarAlarmValue ( rtc_alarm_value_t alarm_val );
```

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Argument(s)]

I/O	Argument	Description
I	`rtc_alarm_value_t alarm_val;`	Alarm conditions (second, minute, hour, weekday, day, month, year)

[Return value]

None.

R_RTC_Set_BinaryAlarmValue

Sets the alarm conditions.(in the case of the binary mode setting)

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`uint32_t alarm_enable;`	Alarm enable (Set the value to the Binary Counter Alarm Enable Register)
	`uint32_t alarm_val`	Alarm conditions (count value) (Set the value to the Binary Counter Alarm Register)

[Return value]

None.

R_RTC_Get_AlarmValue

Reads the alarm conditions (weekday, hour, minute).

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Get_AlarmValue ( rtc_alarm_value_t * const alarm_val );
```

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	`rtc_alarm_value_t * const alarm_val;`	Pointer to structure in which to store the conditions being read

[Return value]

None.

R_RTC_Get_CalendarAlarmValue

Reads the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Get_CalendarAlarmValue ( rtc_alarm_value_t * const alarm_val );
```

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	`rtc_alarm_value_t * const alarm_val;`	Pointer to structure in which to store the conditions being read

[Return value]

None.

R_RTC_Get_BinaryAlarmValue

Reads the alarm conditions (weekday, hour, minute).(in the case of the binary mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Get_BinaryAlarmValue ( uint32_t * const alarm_enable, uint32_t * const
alarm_val );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const alarm_enable`	Pointer to structure in which to store the alarm enable value being read
O	`uint32_t * const alarm_val`	Pointer to structure in which to store the conditions being read

[Return value]

None.

r_rtc_callback_alarm

Performs processing in response to the alarm interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the alarm interrupt INTRTC.

[Syntax]

```
static  void    r_rtc_callback_alarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOn

Enables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Syntax]

```
void    R_RTC_Set_RTC1HZOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOff

Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Syntax]

```
void    R_RTC_Set_RTC1HZOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOn

Enables output of the RTCOUT.

[Syntax]

```
void    R_RTC_Set_RTCOUTOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOff

Disables output of the RTCOUT.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_rtc_alarminterrupt

Performs processing in response to the alarm interrupt INTRTCALM.

Remark This API function is called as the interrupt process corresponding to the alarm interrupt INTRTCALM.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_rtc_alarminterrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_rtc_alarminterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_periodicinterrupt

Performs processing in response to the periodic interrupt INTRTCPRD.

Remark This API function is called as the interrupt process corresponding to the periodic interrupt INTRTCPRD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_rtc_periodicinterrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_rtc_periodicinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_callback_periodic

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the cyclic interrupt INTRTC.

[Syntax]

```
static  void    r_rtc_callback_periodic ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.14 Subsystem clock frequency measurement circuit

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

Table 3.14 API Functions: [Subsystem Clock Frequency Measurement Circuit]

API Function Name	Function
R_FMC_Create	Performs initialization necessary to control the subsystem clock frequency measurement circuit.
R_FMC_Create_UserInit	Performs user-defined initialization relating to the subsystem clock frequency measurement circuit.
r_fmc_interrupt	Performs processing in response to the end of frequency measurement interrupt INTFM.
R_FMC_Start	Starts measurement of the frequency that uses the subsystem clock frequency measurement circuit.
R_FMC_Stop	Ends measurement of the frequency that uses the subsystem clock frequency measurement circuit.
R_FMC_Set_PowerOff	Halts the clock supplied to the subsystem clock frequency measurement circuit.

R_FMC_Create

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

[Syntax]

```
void    R_FMC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Create_UserInit

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

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

[Syntax]

```
void    R_FMC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_fmc_interrupt

Performs processing in response to the end of frequency measurement interrupt INTFM.

Remark This API function is called as the interrupt process corresponding to the end of frequency measurement interrupt INTFM.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_fmc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_fmc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Start

Starts measurement of the frequency that uses the subsystem clock frequency measurement circuit.

[Syntax]

```
void    R_FMC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Stop

Ends measurement of the frequency that uses the subsystem clock frequency measurement circuit.

[Syntax]

```
void    R_FMC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Set_PowerOff

Halts the clock supplied to the subsystem clock frequency measurement circuit.

Remark Calling this API function changes the subsystem clock frequency measurement circuit to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_FMC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.15 12-bit interval timer

Below is a list of API functions output by the Code Generator for 12-bit interval timer use.

Table 3.15 API Functions: [12-Bit Interval Timer]

API Function Name	Function
R_IT_Create	Performs initialization necessary to control the 12-bit interval timer.
R_IT_Create_UserInit	Performs user-defined initialization relating to the 12-bit interval timer.
r_it_interrupt	Performs processing in response to the 12-bit interval timer interrupt INTIT.
R_IT_Start	Starts the count of the 12-bit interval timer.
R_IT_Stop	Ends the count of the 12-bit interval timer.
R_IT_Reset	Reset the 12-bit interval timer.
R_IT_Set_PowerOff	Halts the clock supplied to the 12-bit interval timer.

R_IT_Create

Performs initialization necessary to control the 12-bit interval timer.

[Syntax]

```
void    R_IT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Create_UserInit

Performs user-defined initialization relating to the 12-bit interval timer.

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

[Syntax]

```
void    R_IT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_it_interrupt

Performs processing in response to the 12-bit interval timer interrupt INTIT.

Remark This API function is called as the interrupt process corresponding to the 12-bit interval timer interrupt INTIT.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_it_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_it_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Start

Starts the count of the 12-bit interval timer.

[Syntax]

```
void    R_IT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Stop

Ends the count of the 12-bit interval timer.

[Syntax]

```
void    R_IT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Reset

Reset the 12-bit interval timer.

[Syntax]

```
void    R_IT_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Set_PowerOff

Halts the clock supplied to the 12-bit interval timer.

Remarks 1. Calling this API function changes the 12-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remarks 2. This API function stops the clock supply to the 12-bit interval timer, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-timer clock).

[Syntax]

```
void    R_IT_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.16 8-bit interval timer

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

Table 3.16 API Functions: [8-Bit Interval Timer]

API Function Name	Function
R_IT8bitm_Channeln_Create	Performs initialization necessary to control the 8-bit interval timer.
R_IT8bitm_Channeln_Create_UserInit	Performs user-defined initialization relating to the 8-bit interval timer.
r_it8bitm_channeln_interrupt	Performs processing in response to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.
R_IT8bitm_Channeln_Start	Starts the count of the 8-bit interval timer.
R_IT8bitm_Channeln_Stop	Ends the count of the 8-bit interval timer.
R_IT8bitm_Channeln_Set_PowerOff	Halts the clock supplied to the 8-bit interval timer.
R_IT8bitm_Set_PowerOff	Halts the clock supplied to the 8-bit interval timer.

R_IT8bit*m*_Channel*n*_Create

Performs initialization necessary to control the 8-bit interval timer.

[Syntax]

```
void    R_IT8bitm_Channeln_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8bit*m*_Channel*n*_Create_UserInit

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

Remark This API function is called as the R_IT8bitm_Channeln_Create callback routine.

[Syntax]

```
void    R_IT8bitm_Channeln_Create_UserInit ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_it8bit*m*_channel*n*_interrupt

Performs processing in response to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.

Remark This API function is called as the interrupt process corresponding to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_it8bitm_channeln_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_it8bitm_channeln_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT8bit*m*_Channel*n*_Start

Starts the count of the 8-bit interval timer.

[Syntax]

```
void    R_IT8bitm_Channeln_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8bit*m*_Channel*n*_Stop

Ends the count of the 8-bit interval timer.

[Syntax]

```
void    R_IT8bitm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8bit*m*_Channel*n*_Set_PowerOff

Halts the clock supplied to the 8-bit interval timer.

Remark Calling this API function changes the 8-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_IT8bitm_Channeln_Set_PowerOff ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8bit*m*_Set_PowerOff

Halts the clock supplied to the 8-bit interval timer.

Remark Calling this API function changes the 8-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_IT8bitm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.17 16-bit wakeup timer

Below is a list of API functions output by the Code Generator for 16-bit wakeup timer (WUTM) use.

Table 3.17 API Functions: [16-bit Wakeup Timer]

API Function Name	Function
R_WUTM_Create	Performs initialization necessary to control the 16-bit wakeup timer.
R_WUTM_Create_UserInit	Performs user-defined initialization relating to the 16-bit wakeup timer.
r_wutm_interrupt	Performs processing in response to the timer interrupt.
R_WUTM_Start	Starts the count for 16-bit wakeup timer.
R_WUTM_Stop	Ends the count for 16-bit wakeup timer.
R_WUTM_Set_PowerOff	Halts the clock supplied to the 16-bit wakeup timer.

R_WUTM_Create

Performs initialization necessary to control the 16-bit wakeup timer.

[Syntax]

```
void    R_WUTM_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Create_UserInit

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

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

[Syntax]

```
void    R_WUTM_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wutm_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_wutm_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_wutm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Start

Starts the count for 16-bit wakeup timer.

[Syntax]

```
void    R_WUTM_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Stop

Ends the count for 16-bit wakeup timer.

[Syntax]

```
void    R_WUTM_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Set_PowerOff

Halts the clock supplied to the 16-bit wakeup timer.

Remark Calling this API function changes the 16-bit wakeup timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_WUTM_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.18 Clock output/buzzer output controller

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

Table 3.18 API Functions: [Clock Output/Buzzer Output Controller]

API Function Name	Function
R_PCLBUZn_Create	Performs initialization necessary to control the clock/buzzer output controller.
R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/buzzer output controller.
R_PCLBUZn_Start	Starts clock/buzzer output.
R_PCLBUZn_Stop	Ends clock/buzzer output.
R_PCLBUZ_Set_PowerOff	Halts the clock supplied to the clock/buzzer output controller.

R_PCLBUZ*n*_Create

Performs initialization necessary to control the clock/buzzer output controller.

[Syntax]

```
void    R_PCLBUZn_Create ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Create_UserInit

Performs user-defined initialization relating to the clock/buzzer output controller.

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

[Syntax]

```
void    R_PCLBUZn_Create_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Start

Starts clock/buzzer output.

[Syntax]

```
void    R_PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Stop

Ends clock/buzzer output.

[Syntax]

```
void    R_PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ_Set_PowerOff

Halts the clock supplied to the clock/buzzer output controller.

Remarks 1. Calling this API function changes the clock/buzzer output controller to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remarks 2. This API function stops the clock supply to the clock/buzzer output controller, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

[Syntax]

```
void    R_PCLBUZ_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.19 Watchdog timer

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

Table 3.19 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_interrupt	Performs processing in response to the interval interrupt INTWDTI.
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_interrupt

Performs processing in response to the interval interrupt INTWDTI.

Remark This API function is called as the interrupt process corresponding to the interval interrupt INTWDTI.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_wdt_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_wdt_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.

3.2.20 24-bit DS A/D converter with programmable gain instrumentation amplifier

Below is a list of API functions output by the Code Generator for 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier use.

Table 3.20 API Functions: [24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier]

API Function Name	Function
R_PGA_DSAD_Create	Performs initialization necessary to control the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
R_PGA_DSAD_Create_UserInit	Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
r_pga_dsad_interrupt_conversion	Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.
r_pga_dsad_interrupt_scan	Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.
R_PGA_DSAD_Start	Starts A/D conversion.
R_PGA_DSAD_Stop	Ends A/D conversion.
R_PGA_DSAD_Set_PowerOff	Halts the clock supplied to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
R_PGA_DSAD_Get_AverageResult	Reads the results of A/D conversion.(mean value)
R_PGA_DSAD_Get_Result	Reads the results of A/D conversion.
R_PGA_DSAD_CAMP_OffsetTrimming	Connects the configurable amplifier to the 24-bit ΔΣ A/D converter and trims the offset.
r_pga_dsad_conversion_interrupt	Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.
r_pga_dsad_scan_interrupt	Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

R_PGA_DSAD_Create

Performs initialization necessary to control the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

[Syntax]

```
void    R_PGA_DSAD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Create_UserInit

Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

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

[Syntax]

```
void    R_PGA_DSAD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_interrupt_conversion

Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_pga_dsad_interrupt_conversion ( void );
```

CC-RL Compiler

```
static  void  __near r_pga_dsad_interrupt_conversion ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_interrupt_scan

Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_pga_dsad_interrupt_scan ( void );
```

CC-RL Compiler

```
static  void  __near r_pga_dsad_interrupt_scan ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Start

Starts A/D conversion.

[Syntax]

```
void    R_PGA_DSAD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Stop

Ends A/D conversion.

[Syntax]

```
void    R_PGA_DSAD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Set_PowerOff

Halts the clock supplied to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

[Syntax]

```
void    R_PGA_DSAD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Get_AverageResult

Reads the average results of A/D conversion.(mean value)

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_PGA_DSAD_Get_AverageResult ( uint16_t * const bufferH, uint16_t * const
bufferL );
```

[Argument(s)]

I/O	Argument	Description
O	`uint16_t * const bufferH;`	Pointer to area in which to store read results of A/D conversion (DSADMVM resister and DSADMVH resister)
O	`uint16_t * const bufferL;`	Pointer to area in which to store read results of A/D conversion (DSADMVC resister and DSADMVL resister)

[Return value]

None.

R_PGA_DSAD_Get_Result

Reads the results of A/D conversion.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_PGA_DSAD_Get_Result ( uint16_t * const bufferH, uint16_t * const bufferL );
```

[Argument(s)]

I/O	Argument	Description
O	`uint16_t * const bufferH;`	Pointer to area in which to store the results of A/D conversion (DSADCRM resister and DSADCRH resister)
O	`uint16_t * const bufferL;`	Pointer to area in which to store the results of A/D conversion (DSADCRC resister and DSADCRL resister)

[Return value]

None.

R_PGA_DSAD_CAMP_OffsetTrimming

Connects the configurable amplifier to the 24-bit ΔΣ A/D converter and trims the offset.

[Syntax]

```
void    R_PGA_DSAD_CAMP_OffsetTrimming ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_conversion_interrupt

Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_pga_dsad_conversion_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_pga_dsad_conversion_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_scan_interrupt

Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_pga_dsad_scan_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_pga_dsad_scan_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.21 A/D converter

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

Table 3.21 API Functions: [A/D Converter]

API Function Name	Function
R_ADC_Create	Performs initialization necessary to control the A/D converter.
R_ADC_Create_UserInit	Performs user-defined initialization relating to the A/D converter.
r_adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
R_ADC_Set_OperationOn	Enables operation of voltage converter.
R_ADC_Set_OperationOff	Disables operation of voltage converter.
R_ADC_Start	Starts A/D conversion.
R_ADC_Stop	Ends A/D conversion.
R_ADC_Reset	Reset A/D conversion.
R_ADC_Set_PowerOff	Halts the clock supplied to the A/D converter.
R_ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
R_ADC_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_TestChannel	Sets the operation mode of A/D converter.
R_ADC_Get_Result	Reads the results of A/D conversion (10 bits).
R_ADC_Get_Result_8bit	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

R_ADC_Create

Performs initialization necessary to control the A/D converter.

[Syntax]

```
void    R_ADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Create_UserInit

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

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

[Syntax]

```
void    R_ADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_adc_interrupt

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

Remark This API function is called as the interrupt process corresponding to the A/D conversion end interrupt INTAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_adc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_adc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOn

Enables operation of voltage converter.

Remarks 1. About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to R_ADC_Start.

Remarks 2. On the [A/D Converter], in the [Comparator operation setting] area, if "Operation" is selected, then the voltage converter will be switched to "always on".
There is thus no need to call this API function in this case.

[Syntax]

```
void    R_ADC_Set_OperationOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOff

Disables operation of voltage converter.

[Syntax]

```
void    R_ADC_Set_OperationOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to R_ADC_Set_OperationOn and the call to this API function.

[Syntax]

```
void    R_ADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call R_ADC_Set_OperationOff after the process of this API function completes.

[Syntax]

```
void    R_ADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Reset

Reset A/D conversion.

[Syntax]

```
void    R_ADC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_PowerOff

Halts the clock supplied to the A/D converter.

Remark Calling this API function changes the A/D converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_ADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in argument *channel* is set to analog input channel specification register (ADS).

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_adc.h"
MD_STATUS   R_ADC_Set_ADChannel ( ad_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	`ad_channel_t channel;`	Analog voltage input pin ADCHANNEL*n*: Input pin

Remark See the header file r_cg_adc.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void    R_ADC_Set_SnoozeOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void    R_ADC_Set_SnoozeOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_TestChannel

Sets the operation mode of A/D converter.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_adc.h"
MD_STATUS   R_ADC_Set_TestChannel ( test_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	`test_channel_t channel;`	Operation mode of A/D converter ADNORMALINPUT: Normal mode (Normal A/D conversion) ADAVREFM: Test mode (AVREFM input) ADAVREFP: Test mode (AVREFP input)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Get_Result

Reads the results of A/D conversion (10 bits).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_ADC_Get_Result ( uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
O	`uint16_t  * const buffer;`	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

R_ADC_Get_Result_8bit

Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

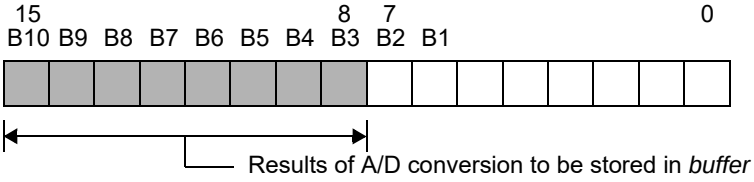
[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_ADC_Get_Result_8bit ( uint8_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const buffer;`	Pointer to area in which to store the results of A/D conversion

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

None.

3.2.22 Configurable amplifier

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

Table 3.22 API Functions: [Configurable amplifier]

API Function Name	Function
R_CAMP_Create	Performs initialization necessary to control the configurable amplifier.
R_CAMP_Create_UserInit	Performs user-defined initialization relating to the configurable amplifier.
R_CAMPn_Start	Turns on the power of the configurable amplifier(AMPn).
R_CAMPn_Stop	Turns off the power of the configurable amplifier(AMPn).
R_CAMP_Set_PowerOff	Halts the clock supplied to the configurable amplifier.

R_CAMP_Create

Performs initialization necessary to control the configurable amplifier.

[Syntax]

```
void    R_CAMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAMP_Create_UserInit

Performs user-defined initialization relating to the configurable amplifier.

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

[Syntax]

```
void    R_CAMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAMP*n*_Start

Turns on the power of the configurable amplifier(AMPn).

[Syntax]

```
void    R_CAMPn_Start ( void );
```

Remark *n* is the channel number.ye

[Argument(s)]

None.

[Return value]

None.

R_CAMP*n*_Stop

Turns off the power of the configurable amplifier(AMPn).

[Syntax]

```
void    R_CAMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CAMP_Set_PowerOff

Halts the clock supplied to the configurable amplifier.

[Syntax]

```
void    R_CAMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.23 Temperature sensor

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

Table 3.23 API Functions: [Temperature Sensor]

API Function Name	Function
R_TMPS_Create	Performs initialization necessary to control the temperature sensor.
R_TMPS_Create_UserInit	Performs user-defined initialization relating to the temperature sensor.
R_TMPS_Start	Starts measurement of the temperature that uses the temperature sensor.
R_TMPS_Stop	Ends measurement of the temperature that uses the temperature sensor.
R_TMPS_Reset	Reset the temperature sensor.
R_TMPS_Set_PowerOff	Halts the clock supplied to the temperature sensor.

R_TMPS_Create

Performs initialization necessary to control the temperature sensor.

[Syntax]

```
void    R_TMPS_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Create_UserInit

Performs user-defined initialization relating to the temperature sensor.

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

[Syntax]

```
void    R_TMPS_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Start

Starts measurement of the temperature that uses the temperature sensor.

[Syntax]

```
void    R_TMPS_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Stop

Ends measurement of the temperature that uses the temperature sensor.

[Syntax]

```
void    R_TMPS_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Reset

Reset the temperature sensor.

[Syntax]

```
void    R_TMPS_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Set_PowerOff

Halts the clock supplied to the temperature sensor.

Remark Calling this API function changes the temperature sensor to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMPS_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.24 24-bit DS A/D converter

Below is a list of API functions output by the Code Generator for 24-bit ΔΣ A/D converter use.

Table 3.24 API Functions: [24-bit ΔΣ A/D Converter]

API Function Name	Function
R_DSADC_Create	Performs initialization necessary to control the 24-bit ΔΣ A/D converter.
R_DSADC_Create_UserInit	Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter.
r_dsadc_interrupt	Performs processing in response to the ΔΣ A/D conversion end interrupt INTDSAD.
r_dsadzcn_interrupt	Performs processing in response to the zero-cross detection interrupt INTDSADZCn.
R_DSADC_Set_OperationOn	Enables operation of 24-bit ΔΣ A/D converter.
R_DSADC_Set_OperationOff	Disables operation of 24-bit ΔΣ A/D converter.
R_DSADC_Start	Starts A/D conversion.
R_DSADC_Stop	Ends A/D conversion.
R_DSADC_Reset	Reset the 24-bit ΔΣ A/D converter.
R_DSADC_Set_PowerOff	Performs electric charge reset for the 24-bit ΔΣ A/D converter.
R_DSADC_ChannelN_Get_Result	Reads the results of A/D conversion (24 bits).
R_DSADC_ChannelN_Get_Result_16bit	Reads the results of A/D conversion (16 bits; most significant 16 bits of 24-bit resolution).

R_DSADC_Create

Performs initialization necessary to control the 24-bit ΔΣ A/D converter.

[Syntax]

```
void    R_DSADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Create_UserInit

Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter.

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

[Syntax]

```
void    R_DSADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsadc_interrupt

Performs processing in response to the ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dsadc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_dsadc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsadzcn_interrupt

Performs processing in response to the zero-cross detection interrupt INTDSADZCn.

Remark This API function is called as the interrupt process corresponding to the zero-cross detection interrupt INTDSADZCn.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dsadzcn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_dsadzcn_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_OperationOn

Enables operation of 24-bit ΔΣ A/D converter.

[Syntax]

```
void    R_DSADC_Set_OperationOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_OperationOff

Disables operation of 24-bit ΔΣ A/D converter.

[Syntax]

```
void    R_DSADC_Set_OperationOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Start

Starts A/D conversion.

[Syntax]

```
void    R_DSADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Stop

Ends A/D conversion.

[Syntax]

```
void    R_DSADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Reset

Reset the 24-bit ΔΣ A/D converter.

[Syntax]

```
void    R_DSADC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_PowerOff

Performs electric charge reset for the 24-bit ΔΣ A/D converter.

Remark About 1.4 microseconds of stabilization time is required when electric charge reset is performed for the 24-bit ΔΣ A/D converter.

[Syntax]

```
void    R_DSADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Channel*n*_Get_Result

Reads the results of A/D conversion (24 bits).

Remark The result of A/D conversion (24 bits) by this API function must be read within the maximum pending time of the ΔΣ A/D conversion result register n after ΔΣ A/D conversion end interrupt INTDSAD is generated.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DSADC_Channeln_Get_Result ( uint32_t * const buffer );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const buffer;`	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

R_DSADC_Channel*n*_Get_Result_16bit

Reads the results of A/D conversion (16 bits; most significant 16 bits of 24-bit resolution).

Remark The result of A/D conversion by this API function must be read within the maximum pending time of the ΔΣ A/D conversion result register n after ΔΣ A/D conversion end interrupt INTDSAD is generated.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DSADC_Channeln_Get_Result_16bit ( uint16_t * const buffer );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint16_t * const buffer;`	Pointer to area in which to store the results of A/D conversion

[Return value]

None.

3.2.25 D/A converter

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

Table 3.25 API Functions: [D/A Converter]

API Function Name	Function
R_DAC_Create	Performs initialization necessary to control the D/A converter.
R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DACn_Start	Starts D/A conversion.
R_DACn_Stop	Ends D/A conversion.
R_DAC_Set_PowerOff	Halts the clock supplied to the D/A converter.
R_DACn_Set_ConversionValue	Sets the analog voltage output to the ANO*n* pin.
R_DAC_Change_OutputVoltage_8bit	Changes the output voltage of D/A converter.(8bit mode)
R_DAC_Change_OutputVoltage	Changes the output voltage of D/A converter.(12bit mode)
R_DACn_Create	Performs initialization necessary to control the D/A converter.
R_DAC_Reset	Reset the D/A converter.
R_DACn_Create_UserInit	Performs user-defined initialization relating to the D/A converter.

R_DAC_Create

Performs initialization necessary to control the D/A converter.

[Syntax]

```
void    R_DAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC_Create_UserInit

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

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

[Syntax]

```
void    R_DAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Start

Starts D/A conversion.

[Syntax]

```
void    R_DACn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Stop

Ends D/A conversion.

[Syntax]

```
void    R_DACn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Set_PowerOff

Halts the clock supplied to the D/A converter.

Remark Calling this API function changes the D/A converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_DAC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Set_ConversionValue

Sets the analog voltage output to the ANO*n* pin.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DACn_Set_ConversionValue ( uint8_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t reg_value;`	D/A conversion value (0x0 to 0xFF)

[Return value]

None.

R_DAC_Change_OutputVoltage_8bit

Changes the output voltage of D/A converter.(8bit mode)

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DAC_Change_OutputVoltage_8bit ( uint8_t outputVoltage );
```

[Argument(s)]

I/O	Argument	Description
I	`uint8_t outputVoltage;`	`output Voltage (Low 8bit)`

[Return value]

None.

R_DAC_Change_OutputVoltage

Changes the output voltage of D/A converter.(12bit mode)

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DAC_Change_OutputVoltage ( uint16_t outputVoltage );
```

[Argument(s)]

I/O	Argument	Description
I	`uint16_t outputVoltage;`	`output Voltage (Low 12bit)`

[Return value]

None.

R_DAC*n*_Create

Performs initialization necessary to control the D/A converter.

[Syntax]

```
#include "r_cg_macrodriver.h"
void    R_DACn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Reset

Reset the D/A converter.

[Syntax]

```
void    R_DAC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Create_UserInit

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

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

[Syntax]

```
void    R_DACn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.26 Programmable gain amplifier

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

Table 3.26 API Functions: [Programmable Gain Amplifier]

API Function Name	Function
R_PGA_Create	Performs initialization necessary to control the programmable gain amplifier.
R_PGA_Create_UserInit	Performs user-defined initialization relating to the programmable gain amplifier.
R_PGA_Start	Starts the operation of programmable gain amplifier.
R_PGA_Stop	Ends the operation of programmable gain amplifier.
R_PGA_Reset	Reset the programmable gain amplifier.
R_PGA_Set_PowerOff	Halts the clock supplied to the programmable gain amplifier.

R_PGA_Create

Performs initialization necessary to control the programmable gain amplifier.

[Syntax]

```
void    R_PGA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Create_UserInit

Performs user-defined initialization relating to the programmable gain amplifier.

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

[Syntax]

```
void    R_PGA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Start

Starts the operation of programmable gain amplifier.

[Syntax]

```
void    R_PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Stop

Ends the operation of programmable gain amplifier.

[Syntax]

```
void    R_PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Reset

Reset the operation of programmable gain amplifier.

[Syntax]

```
void    R_PGA_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Set_PowerOff

Halts the clock supplied to the programmable gain amplifier.

Remark Calling this API function changes the comparator to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_PGA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.27 Comparator

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

Table 3.27 API Functions: [Comparator]

API Function Name	Function
R_COMP_Create	Performs initialization necessary to control the comparator.
R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
r_compn_interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
R_COMP_Reset	Reset the comparator.
R_COMP_Set_PowerOff	Halts the clock supplied to the comparator.

R_COMP_Create

Performs initialization necessary to control the comparator.

[Syntax]

```
void    R_COMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Create_UserInit

Performs user-defined initialization relating to the comparator.

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

[Syntax]

```
void    R_COMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_comp*n*_interrupt

Performs processing in response to the comparator interrupt INTCMP*n*.

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_compn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_compn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Start

Begins comparison of reference input voltage and analog input voltage.

[Syntax]

```
void    R_COMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Stop

Stops comparison of reference input voltage and analog input voltage.

[Syntax]

```
void    R_COMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP_Reset

Reset the comparator.

[Syntax]

```
void    R_COMP_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Set_PowerOff

Halts the clock supplied to the comparator.

Remark Calling this API function changes the comparator to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_COMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.28 Comparator/ProgrammableGainAmplifier

Below is a list of API functions output by the Code Generator for comparator/programmable gain amplifier use.

Table 3.28 API Functions: [Comparator/ProgrammableGainAmplifier]

API Function Name	Function
R_COMPPGA_Create	Performs initialization necessary to control the comparator/programmable gain amplifier.
R_COMPPGA_Set_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifier.
R_COMPPGA_Create_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifier.
r_compn_interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
R_PGA_Start	Starts the operation of programmable gain amplifier.
R_PGA_Stop	Ends the operation of programmable gain amplifier.
R_PWMOPT_Start	Supplies the clock to the 6-phase PWM option. In addition, sets the operation mode of the 6-phase PWM option.
R_PWMOPT_Stop	Halts the clock supplied to the 6-phase PWM option.

R_COMPPGA_Create

Performs initialization necessary to control the comparator/programmable gain amplifier.

[Syntax]

```
void    R_COMPPGA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMPPGA_Set_PowerOff

Halts the clock supplied to the comparator/programmable gain amplifier.

Remark Calling this API function changes the comparator/programmable gain amplifier to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_COMPPGA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMPPGA_Create_UserInit

Performs user-defined initialization relating to the comparator/programmable gain amplifier.

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

[Syntax]

```
void    R_COMPPGA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_comp*n*_interrupt

Performs processing in response to the comparator interrupt INTCMP*n*.

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_compn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_compn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Start

Begins comparison of reference input voltage and analog input voltage.

[Syntax]

```
void    R_COMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Stop

Stops comparison of reference input voltage and analog input voltage.

[Syntax]

```
void    R_COMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_PGA_Start

Starts the operation of programmable gain amplifier.

[Syntax]

```
void    R_PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Stop

Ends the operation of programmable gain amplifier.

[Syntax]

```
void    R_PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PWMOPT_Start

Supplies the clock to the 6-phase PWM option.
In addition, sets the operation mode of the 6-phase PWM option.

[Syntax]

```
void    R_PWMOPT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PWMOPT_Stop

Halts the clock supplied to the 6-phase PWM option.

[Syntax]

```
void    R_PWMOPT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.29 Serial array unit

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

Table 3.29 API Functions: [Serial Array Unit]

API Function Name	Function
R_SAUm_Create	Performs initialization necessary to control the serial array unit.
R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit.
R_SAUm_Reset	Reset the serial array unit.
R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
R_SAUm_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_SAUm_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_UARTn_Create	Performs initialization necessary to perform the UART communication.
r_uartn_interrupt_send	Performs processing in response to the UART transmission end interrupt INTST*n*.
r_uartn_interrupt_receive	Performs processing in response to the UART reception end interrupt INTSR*n*.
r_uartn_interrupt_error	Performs processing in response to the reception error interrupt INTSRE*n*.
R_UARTn_Start	Sets UART communication to standby mode.
R_UARTn_Stop	Ends UART communication.
R_UARTn_Send	Starts UART data transmission.
R_UARTn_Receive	Starts UART data reception.
r_uartn_callback_sendend	Performs processing in response to the UART transmission end interrupt INTST*n*.
r_uartn_callback_receiveend	Performs processing in response to the UART reception end interrupt INTSR*n*.
r_uartn_callback_error	Performs processing in response to the UART reception error interrupt INTSRE*n*.
r_uartn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
R_CSImn_Create	Performs initialization necessary to perform the 3-wire serial I/O communication.
r_csimn_interrupt	Performs processing in response to the CSI communication end interrupt INTCSI*mn*.
R_CSImn_Start	Sets 3-wire serial I/O communication to standby mode.
R_CSImn_Stop	Ends 3-wire serial I/O communication.
R_CSImn_Send	Starts CSI data transmission.
R_CSImn_Receive	Starts CSI data reception.
R_CSImn_Send_Receive	Starts CSI data transmission/reception.
r_csimn_callback_sendend	Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

API Function Name	Function
r_csimn_callback_receiveend	Performs processing in response to the CSI reception end interrupt INTCSI*mn*.
r_csimn_callback_error	Performs processing in response to the CSI reception error interrupt INTSRE*n*.
R_IICmn_Create	Performs initialization necessary to perform the simplified IIC communication.
r_iicmn_interrupt	Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.
R_IICmn_StartCondition	Generates start conditions.
R_IICmn_StopCondition	Generates stop conditions.
R_IICmn_Stop	Ends simplified IIC communication.
R_IICmn_Master_Send	Starts simple IIC master transmission.
R_IICmn_Master_Receive	Starts simple IIC master reception.
r_iicmn_callback_master_sendend	Performs processing in response to the simple IIC*mn* master transmission end interrupt INTIIC*mn*.
r_iicmn_callback_master_receiveend	Performs processing in response to the simple IIC*mn* master reception end interrupt INTIIC*mn*.
r_iicmn_callback_master_error	Performs processing in response to detection of parity error (ACK error).

R_SAU*m*_Create

Performs initialization necessary to control the serial array unit.

[Syntax]

```
void    R_SAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Create_UserInit

Performs user-defined initialization related to the serial array unit.

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

[Syntax]

```
void    R_SAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Reset

Reset the serial array unit.

[Syntax]

```
void    R_SAUm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_PowerOff

Halts the clock supplied to the serial array unit.

Remark Calling this API function changes the serial array unit to reset status.
For this reason, writes to the control registers (e.g. serial clock select register *n*: SPS*n*) after this API function is called are ignored.

[Syntax]

```
void    R_SAUm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void    R_SAUm_Set_SnoozeOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void    R_SAUm_Set_SnoozeOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Create

Performs initialization necessary to perform the UART communication.

Remark This API function is used as an internal function of R_SAUm_Create.
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void    R_UARTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_send

Performs processing in response to the UART transmission end interrupt INTST*n*.

Remark This API function is called as the interrupt process corresponding to the UART transmission end interrupt INTST*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartn_interrupt_send ( void );
```

CC-RL Compiler

```
static  void  __near r_uartn_interrupt_send ( void );
```

Remark [*n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_receive

Performs processing in response to the UART reception end interrupt INTSR*n*.

Remark This API function is called as the interrupt process corresponding to the UART reception end interrupt INTSR*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartn_interrupt_receive ( void );
```

CC-RL Compiler

```
static  void  __near r_uartn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_error

Performs processing in response to the reception error interrupt INTSRE*n*.

Remark This API function is called as the interrupt process corresponding to the reception error interrupt INTSRE*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartn_interrupt_error ( void );
```

CC-RL Compiler

```
static  void  __near r_uartn_interrupt_error ( void );
```

Remark [*n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Start

Sets UART communication to standby mode.

[Syntax]

```
void    R_UARTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Stop

Ends UART communication.

[Syntax]

```
void    R_UARTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Send

Starts UART data transmission.

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

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTn_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 specification

R_UART*n*_Receive

Starts UART data reception.

Remarks 1. This API function performs byte-level UART 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. Actual UART reception starts after this API function is called, and R_UARTn_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTn_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 received data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_uart*n*_callback_sendend

Performs processing in response to the UART transmission end interrupt INTST*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_send corresponding to the UART transmission end interrupt INTST*n* (performed when number of transmission data specified by R_UARTn_Send argument *tx_num* has been completed).

[Syntax]

```
static  void    r_uartn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_callback_receiveend

Performs processing in response to the UART reception end interrupt INTSR*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_receive corresponding to the UART reception end interrupt INTSR*n* (performed when number of received data specified by R_UARTn_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_uartn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_callback_error

Performs processing in response to the UART reception error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_error corresponding to the UART reception error interrupt INTSRE*n*.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_uartn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for UART reception error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing error

[Return value]

None.

r_uart*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_receive corresponding to the UART reception end interrupt INTSR*n* (process performed when the amount of data received is greater than the argument *rx_num* specified for R_UARTn_Receive).

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_uartn_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint16_t  rx_data;`	Receive data (greater than the argument *rx_num* specified for R_UARTn_Receive)

[Return value]

None.

R_CSI*mn*_Create

Performs initialization necessary to perform the 3-wire serial I/O communication.

Remark This API function is used as an internal function of R_SAUm_Create.
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void    R_CSImn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_interrupt

Performs processing in response to the CSI communication end interrupt INTCSI*mn*.

Remark This API function is called as the interrupt process corresponding to the CSI communication end interrupt INTCSI*mn*.

[Syntax]

```
__interrupt static  void    r_csimn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Start

Sets 3-wire serial I/O communication to standby mode.

[Syntax]

```
void    R_CSImn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Stop

Ends 3-wire serial I/O communication.

[Syntax]

```
void    R_CSImn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Send

Starts CSI data transmission.

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

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

[Syntax]

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

Remark *m* is the unit number, and *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 specification

R_CSI*mn*_Receive

Starts CSI data reception.

Remarks 1. This API function performs byte-level CSI 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 CSI reception, R_CSImn_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSI*mn*_Send_Receive

Starts CSI data transmission/reception.

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

Remarks 2. This API function performs byte-level CSI reception the number of times specified by the argument *tx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remarks 3. When performing a CSI reception, R_CSImn_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the unit number, and *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 received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_csi*mn*_callback_sendend

Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process r_csimn_interrupt corresponding to the CSI transmission end interrupt INTCSI*mn* (performed when number of transmission data specified by R_CSImn_Send or R_CSImn_Send_Receive argument *tx_num* has been completed).

[Syntax]

```
static  void    r_csimn_callback_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_callback_receiveend

Performs processing in response to the CSI reception end interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process r_csimn_interrupt corresponding to the CSI reception end interrupt INTCSI*mn* (performed when number of received data specified by R_CSImn_Receive or R_CSImn_Send_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_csimn_callback_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_callback_error

Performs processing in response to the CSI reception error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_error corresponding to the CSI reception error interrupt INTSRE*n*.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_csimn_callback_error ( uint8_t err_type );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for CSI reception error interrupt 00000*xx*1B: Overrun error

[Return value]

None.

R_IIC*mn*_Create

Performs initialization necessary to perform the simplified IIC communication.

Remark This API function is used as an internal function of R_SAUm_Create.
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void    R_IICmn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_interrupt

Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.

Remark This API function is called as the interrupt process corresponding to the simple IIC communication end interrupt INTIIC*mn*.

[Syntax]

```
__interrupt static  void    r_iicmn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StartCondition

Generates start conditions.

Remark This API function is used as an internal function of R_IICmn_Master_Send and R_IICmn_Master_Receive.
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void    R_IIC mn _StartCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StopCondition

Generates stop conditions.

[Syntax]

```
void    R_IICmn_StopCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Stop

Ends simple IIC communication.

[Syntax]

```
void    R_IICmn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Master_Send

Starts simple IIC master transmission.

Remark This API function repeats the byte-level simple IIC master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

[Syntax]

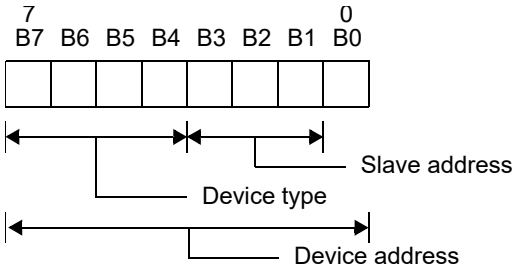
```
#include    "r_cg_macrodriver.h"
void    R_IIC mn _Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t  adr;`	Device 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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

R_IIC*mn*_Master_Receive

Starts simple IIC master reception.

Remark This API function performs byte-level simple IIC master reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Syntax]

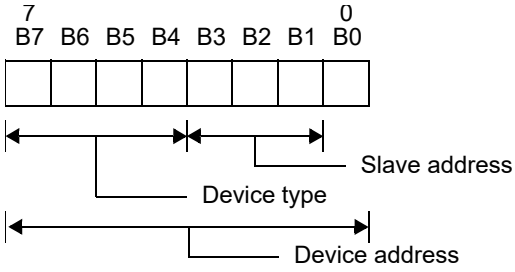
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num
);
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

r_iic*mn*_callback_master_sendend

Performs processing in response to the simple IIC*mn* master transmission end interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process r_iicmn_interrupt corresponding to the simple IIC*mn* master transmission end interrupt INTIIC*mn* (performed when number of transmission data specified by R_IICmn_Master_Send argument *tx_num* has been completed).

[Syntax]

```
static  void    r_iicmn_callback_master_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_callback_master_receiveend

Performs processing in response to the simple IIC*mn* master reception end interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process r_iicmn_interrupt corresponding to the simple IIC*mn* master reception end interrupt INTIIC*mn* (performed when number of received data specified by R_IICmn_Master_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_iicmn_callback_master_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_callback_master_error

Performs processing in response to detection of parity error (ACK error).

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_iicmn_callback_master_error ( MD_STATUS flag );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`MD_STATUS flag;`	Cause of communication error MD_NACK: Acknowledge not detected

[Return value]

None.

3.2.30 Serial array unit 4

Below is a list of API functions output by the Code Generator for serial array unit 4 (DALI/UART4) use.

Table 3.30 API Functions: [Serial Array Unit 4]

API Function Name	Function
R_DALIn_Create	Performs initialization necessary to control the serial array unit 4 (DALI/UART4).
r_dalin_interrupt_send	Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.
r_dalin_interrupt_receive	Performs processing in response to the DALI reception end interrupt INTSRDL*n*.
r_dalin_interrupt_error	Performs processing in response to the DALI reception error interrupt INTSREDL*n*.
R_DALIn_Start	Sets DALI communication to standby mode.
R_DALIn_Stop	Ends DALI communication.
R_DALIn_Send	Starts DALI data transmission.
R_DALIn_Receive	Starts DALI data reception.
r_dalin_callback_sendend	Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.
r_dalin_callback_receiveend	Performs processing in response to the DALI reception end interrupt INTSRDL*n*.
r_dalin_callback_error	Performs processing in response to the DALI reception error interrupt INTSREDL*n*.
r_dalin_callback_softwareoverrun	Performs processing in response to detection of overrun error.

R_DALI*n*_Create

Performs initialization necessary to control the serial array unit 4 (DALI/UART4).

[Syntax]

```
void    R_DALIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_send

Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI transmission end interrupt INTSTDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dalin_interrupt_send ( void );
```

CC-RL Compiler

```
static  void  __near r_dalin_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_receive

Performs processing in response to the DALI reception end interrupt INTSRDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI reception end interrupt INTSRDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dalin_interrupt_receive ( void );
```

CC-RL Compiler

```
static  void  __near r_dalin_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_error

Performs processing in response to the DALI reception error interrupt INTSREDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI reception error interrupt INTSREDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dali n_interrupt_error ( void );
```

CC-RL Compiler

```
static  void  __near r_dali n_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Start

Sets DALI communication to standby mode.

[Syntax]

```
void    R_DALIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Stop

Ends DALI communication.

[Syntax]

```
void    R_DALIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Send

Starts DALI data transmission.

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

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_DALIn_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 specification

R_DALI*n*_Receive

Starts DALI data reception.

Remarks 1. This API function performs byte-level DALI 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. Actual DALI reception starts after this API function is called, and R_DALIn_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_DALIn_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 received data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_dali*n*_callback_sendend

Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_send corresponding to the DALI transmission end interrupt INTSTDL*n* (performed when number of transmission data specified by R_DALIn_Send argument *tx_num* has been completed).

[Syntax]

```
static  void    r_dali n_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_callback_receiveend

Performs processing in response to the DALI reception end interrupt INTSRDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_receive corresponding to the DALI reception end interrupt INTSRDL*n* (performed when number of received data specified by R_DALIn_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_dali n_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_callback_error

Performs processing in response to the DALI reception error interrupt INTSREDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_error corresponding to the DALI reception error interrupt INTSREDL*n*.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_dali n_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for DALI reception error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing error

[Return value]

None.

r_dali*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_receive corresponding to the DALI reception end interrupt INTSRDL*n* (process performed when the amount of data received is greater than the argument *rx_num* specified for R_DALIn_Receive).

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_dali n_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint16_t  rx_data;`	Receive data (greater than the argument *rx_num* specified for R_DALIn_Receive)

[Return value]

None.

3.2.31 Asynchronous serial interface LIN-UART

Below is a list of API functions output by the Code Generator for asynchronous serial interface LIN-UART (UARTF) use.

Table 3.31 API Functions: [Asynchronous Serial Interface LIN-UART]

API Function Name	Function
R_UARTFn_Create	Performs initialization necessary to control the asynchronous serial interface LIN-UART (UARTF).
R_UARTFn_Create_UserInit	Performs user-defined initialization related to the asynchronous serial interface LIN-UART (UARTF).
r_uartfn_interrupt_send	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_interrupt_receive	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_interrupt_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
R_UARTFn_Start	Sets LIN communication to standby mode.
R_UARTFn_Stop	Ends LIN communication.
R_UARTFn_Set_PowerOff	Halts the clock supplied to the asynchronous serial interface LIN-UART (UARTF).
R_UARTFn_Send	Starts UARTF data transmission.
R_UARTFn_Receive	Starts UARTF data reception.
R_UARTFn_Set_DataComparisonOn	Starts the data comparison.
R_UARTFn_Set_DataComparisonOff	Ends the data comparison.
r_uartfn_callback_sendend	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_callback_receiveend	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_callback_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
r_uartfn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
r_uartfn_callback_expbitdetect	Performs processing in response to detection of expansion bit.
r_uartfn_callback_idmatch	Performs processing in response to match of ID parity.

R_UARTF*n*_Create

Performs initialization necessary to control the asynchronous serial interface LIN-UART (UARTF).

[Syntax]

```
void    R_UARTFn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Create_UserInit

Performs user-defined initialization related to the asynchronous serial interface LIN-UART (UARTF).

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

[Syntax]

```
void    R_UARTFn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_send

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the interrupt process corresponding to the LIN-UART transmission end interrupt INTLT.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartfn_interrupt_send ( void );
```

CC-RL Compiler

```
static  void  __near r_uartfn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_receive

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception end interrupt INTLR.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartfn_interrupt_receive ( void );
```

CC-RL Compiler

```
static  void  __near r_uartfn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception status interrupt INTLS.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartfn_interrupt_error ( void );
```

CC-RL Compiler

```
static  void  __near r_uartfn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Start

Sets LIN communication to standby mode.

[Syntax]

```
void    R_UARTFn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Stop

Ends LIN communication.

[Syntax]

```
void    R_UARTFn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Set_PowerOff

Halts the clock supplied to the asynchronous serial interface LIN-UART (UARTF).

Remark Calling this API function changes the asynchronous serial interface LIN-UART (UARTF) to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_UARTFn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Send

Starts UARTF data transmission.

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

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

Remarks 3. If the asynchronous serial interface LIN-UART (UARTF) is used in expansion bit mode, then store the data to send in the buffer specified by argument *tx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Syntax]

```
MD_STATUS   R_UARTFn_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 specification
MD_DATAEXISTS	Executing transmission process

R_UARTF*n*_Receive

Starts UARTF data reception.

Remarks 1. This API function performs byte-level UARTF 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. Actual UARTF reception starts after this API function is called, and R_UARTFn_Start is then called.

Remarks 3. If the asynchronous serial interface LIN-UART (UARTF) is used in expansion bit mode, then the received data is stored in the buffer specified by argument *rx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Syntax]

```
MD_STATUS   R_UARTFn_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 received data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UARTF*n*_Set_DataComparisonOn

Starts the data comparison.

Remark Calling this API function switches the asynchronous serial interface LIN-UART (UARTF) to expansion bit mode (with data comparison).

[Syntax]

```
void    R_UARTFn_Set_DataComparisonOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Set_DataComparisonOff

Ends the data comparison.

Remark Calling this API function switches the asynchronous serial interface LIN-UART (UARTF) to expansion bit mode (with no data comparison).

[Syntax]

```
void    R_UARTFn_Set_DataComparisonOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_sendend

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_send corresponding to the LIN-UART transmission end interrupt INTLT (performed when number of transmission data specified by R_UARTFn_Send argument *tx_num* has been completed).

[Syntax]

```
static  void    r_uartfn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_receiveend

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_receive corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by R_UARTFn_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_uartfn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for LIN-UART reception status interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing error

[Return value]

None.

r_uartf*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_receive corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by R_UARTFn_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_uartfn_callback_softwareoverrun ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_expbitdetect

Performs processing in response to detection of expansion bit.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS (performed when expansion bit has been detected).

[Syntax]

```
static  void    r_uartfn_callback_expbitdetect ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_idmatch

Performs processing in response to match ID parity.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS (performed when ID parity has been matched).

[Syntax]

```
static  void    r_uartfn_callback_idmatch ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.32 Serial interface IICA

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

Table 3.32 API Functions: [Serial Interface IICA]

API Function Name	Function
R_IICAn_Create	Performs initialization necessary to control the serial interface IICA.
R_IICAn_Create_UserInit	Performs user-defined initialization related to the serial interface IICA.
r_iican_interrupt	Performs processing in response to the IICA communication end interrupt INTIICA*n*.
R_IICAn_StopCondition	Generates stop conditions.
R_IICAn_Stop	Ends IICA communication.
R_IICAn_Reset	Reset the serial interface IICA.
R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface IICA.
R_IICAn_Master_Send	Starts IICA master transmission.
R_IICAn_Master_Receive	Starts IICA master reception.
r_iican_callback_master_sendend	Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.
r_iican_callback_master_receiveend	Performs processing in response to the IICA master reception end interrupt INTIICA*n*.
r_iican_callback_master_error	Performs processing in response to detection of IICA master communication error.
R_IICAn_Slave_Send	Starts IICA slave transmission.
R_IICAn_Slave_Receive	Starts IICA slave reception.
r_iican_callback_slave_sendend	Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.
r_iican_callback_slave_receiveend	Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.
r_iican_callback_slave_error	Performs processing in response to detection of IICA slave communication error.
r_iican_callback_getstopcondition	Performs processing in response to detection of stop condition.
R_IICAn_Set_SnoozeOn	Enables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_SnoozeOff	Disables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_WakeupOn	Enables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_WakeupOff	Disables operation of the address match wakeup function in STOP mode.

R_IICA*n*_Create

Performs initialization necessary to control the serial interface IICA.

[Syntax]

```
void    R_IICAn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Create_UserInit

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

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

[Syntax]

```
void    R_IICAn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_interrupt

Performs processing in response to the IICA communication end interrupt INTIICA*n*.

Remark This API function is called as the interrupt process corresponding to the IICA communication end interrupt INTIICA*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_iican_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_iican_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_StopCondition

Generates stop conditions.

Remark After calling this API function, please confirm a detection of stop condition by SPD0 bit before stopping IICA.

[Syntax]

```
void    R_IICAn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Stop

Ends IICA communication.

[Syntax]

```
void    R_IICAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Reset

Reset the serial interface IICA.

[Syntax]

```
void    R_IICAn_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_PowerOff

Halts the clock supplied to the serial interface IICA.

Remark Calling this API function changes the serial interface IICA to reset status.
For this reason, writes to the control registers (e.g. IICA control register *n*: IICCTL*n*) after this API function is called are ignored.

[Syntax]

```
void    R_IICAn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Master_Send

Starts IICA master transmission.

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

[Syntax]

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

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
I	`uint8_t wait;`	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

R_IICA*n*_Master_Receive

Starts IICA master reception.

Remark This API function performs byte-level IICA master reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Syntax]

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

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 received data
I	`uint16_t rx_num;`	Total amount of data to receive
I	`uint8_t wait;`	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

r_iica*n*_callback_master_sendend

Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA master transmission end interrupt INTIICA*n*.

[Syntax]

```
static  void    r_iica n_callback_master_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_master_receiveend

Performs processing in response to the IICA master reception end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA master reception end interrupt INTIICA*n*.

[Syntax]

```
static  void    r_iica n_callback_master_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_master_error

Performs processing in response to detection of IICA master communication error.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_iica n_callback_master_error ( MD_STATUS flag  );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS flag;`	Cause of communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge not detected (No slave that matches the address/A slave can receive no more data or does not require the next data)

[Return value]

None.

R_IICA*n*_Slave_Send

Starts IICA slave transmission.

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_IICAn_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_IICA*n*_Slave_Receive

Starts IICA slave reception.

Remark This API function performs byte-level IICA 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*.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_IICAn_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 received data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

None.

r_iica*n*_callback_slave_sendend

Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA slave transmission end interrupt INTIICA*n*.

[Syntax]

```
static  void    r_iica n_callback_slave_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_slave_receiveend

Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA slave reception end interrupt INTIICA*n*.

[Syntax]

```
static  void    r_iica n_callback_slave_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_slave_error

Performs processing in response to detection of IICA slave communication error.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_iica n_callback_slave_error ( MD_STATUS flag  );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS flag;`	Cause of communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge not detected (Master receiving end)

[Return value]

None.

r_iica*n*_callback_getstopcondition

Performs processing in response to detection of stop condition.

[Syntax]

```
static  void    r_iica n_callback_getstopcondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_SnoozeOn

Enables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void    R_IICAn_Set_SnoozeOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_SnoozeOff

Disables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void    R_IICAn_Set_SnoozeOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_WakeupOn

Enables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void    R_IICAn_Set_WakeupOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_WakeupOff

Disables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void    R_IICAn_Set_WakeupOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.33 LCD controller/driver

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

Table 3.33 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_interrupt	Performs processing in response to the LCD frame interrupt INTLCD.
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_Set_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_VoltageOff	Disables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_PowerOff	Halts the clock supplied to the LCD controller/driver.
R_LCD_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_VoltageOff	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_interrupt

Performs processing in response to the LCD frame interrupt INTLCD.

Remark This API function is called as the interrupt process corresponding to the LCD frame interrupt INTLCD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lcd_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_lcd_interrupt ( 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_Set_VoltageOn

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Set_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_VoltageOff

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Set_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_PowerOff

Halts the clock supplied to the LCD controller/driver.

Remarks 1. Calling this API function changes the LCD controller/driver to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remarks 2. This API function stops the clock supply to the LCD controller/driver, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

[Syntax]

```
void    R_LCD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_VoltageOn

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_VoltageOff

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.34 Sound generator

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

Table 3.34 API Functions: [Sound Generator]

API Function Name	Function
R_SG_Create	Performs initialization necessary to control the sound generator.
R_SG_Create_UserInit	Performs user-defined initialization relating to the sound generator.
r_sg_interrupt	Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.
R_SG_Start	Enables operation of sound generator.
R_SG_Stop	Disables operation of sound generator.

R_SG_Create

Performs initialization necessary to control the sound generator.

[Syntax]

```
void    R_SG_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Create_UserInit

Performs user-defined initialization relating to the sound generator.

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

[Syntax]

```
void    R_SG_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_sg_interrupt

Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.

Remark This API function is called as the interrupt process corresponding to the threshold value detection of the logarithmic decrement interrupt INTSG.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_sg_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_sg_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Start

Enables operation of sound generator.

[Syntax]

```
void    R_SG_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Stop

Disables operation of sound generator.

[Syntax]

```
void    R_SG_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.35 DMA controller

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

Table 3.35 API Functions: [DMA Controller]

API Function Name	Function
R_DMACn_Create	Performs initialization necessary to control the DMA controller.
R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
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_dmacn_interrupt	Performs processing in response to the DMA transfer end interrupt INTDMA*n*.
R_DMACn_Start	Enables operation of channel *n*.
R_DMACn_Stop	Disables operation of channel *n*.
R_DMACn_Set_SoftwareTriggerOn	Starts DMA transfer.

R_DMAC*n*_Create

Performs initialization necessary to control the DMA controller.

[Syntax]

```
void    R_DMACn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Syntax]

```
void    R_DMACn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

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*n*_interrupt

Performs processing in response to the DMA transfer end interrupt INTDMA*n*.

Remark This API function is called as the interrupt process corresponding to the DMA transfer end interrupt INTDMA*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_dmacn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_dmacn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Start

Enables operation of channel *n*.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Stop

Disables operation of channel *n*.

Remarks 1. This API function does not forcibly terminate DMA transfer.

Remarks 2. Before using this API function, you must confirm that transmission has ended.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Set_SoftwareTriggerOn

Starts DMA transfer.

[Syntax]

```
void    R_DMACn_Set_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.36 Data transfer controller

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

Table 3.36 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_DTCn_Start	Enables operation of the Data transfer controller.
R_DTCn_Stop	Disables operation of the Data transfer controller.
R_DTC_Set_PowerOff	Halts the clock supplied to the Data transfer controller.
R_DTCDn_Start	Enables operation of the Data transfer controller.
R_DTCDn_Stop	Disables operation of the Data transfer controller.

R_DTC_Create

Performs initialization necessary to control the DTC.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DTC_Create_UserInit

Performs user-defined initialization relating to the DTC.

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*n*_Start

Enables operation of the DTC.

[Syntax]

```
void    R_DTCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*n*_Stop

Disables operation of the DTC.

[Syntax]

```
void    R_DTCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC_Set_PowerOff

Halts the clock supplied to the DTC.

Remark Calling this API function changes the DTC to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_DTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTCD*n*_Start

Enables operation of the DTC.

[Syntax]

```
void    R_DTCDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTCD*n*_Stop

Disables operation of the DTC.

[Syntax]

```
void    R_DTCDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.37 Event link controller

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

Table 3.37 API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller (ELC).
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller (ELC).
R_ELC_Stop	Disables operation of the event link controller (ELC).

R_ELC_Create

Performs initialization necessary to control the event link controller (ELC).

[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 (ELC).

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_Stop

Disables operation of the event link controller (ELC).

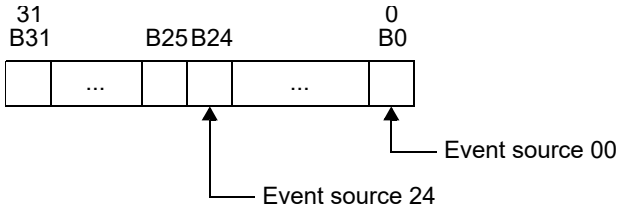
[Syntax]

```
void    R_ELC_Stop ( uint32_t event );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t event;`	Disabled event source

Remark Below is shown the format for specifying disabled event source *event*.
In case of setting the *event* to 0x01010101, the event link operations of event source 00, 08, 16, 24 are prohibited.



[Return value]

None.

3.2.38 Interrupt functions

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

Table 3.38 API Functions: [Interrupt Functions]

API Function Name	Function
R_INTC_Create	Performs initialization necessary to control the interrupt functions.
R_INTC_Create_UserInit	Performs user-defined initialization relating to the interrupt functions.
r_intcn_interrupt	Performs processing in response to the external maskable interrupt INTP*n*.
R_INTCn_Start	Enables the acceptance of the external maskable interrupts INTP*n*.
R_INTCn_Stop	Disables the acceptance of the external maskable interrupts INTP*n*.
r_intclrn_interrupt	Performs processing in response to the external maskable interrupt INTPLR*n*.
R_INTCLRn_Start	Enables the acceptance of the external maskable interrupts INTPLR*n*.
R_INTCLRn_Stop	Disables the acceptance of the external maskable interrupts INTPLR*n*.
r_intrtcicn_interrupt	Performs processing in response to the external maskable interrupt INTRTCIC*n*.
R_INTRTCICn_Start	Enables the acceptance of the external maskable interrupts INTRTCIC*n*.
R_INTRTCICn_Stop	Disables the acceptance of the external maskable interrupts INTRTCIC*n*.
R_INTFO_Start	Enables the acceptance of the external maskable interrupts INTFO.
R_INTFO_Stop	Disables the acceptance of the external maskable interrupts INTFO.
R_INTFO_ClearFlag	Clears INTFCLR flag of Interrupt flag output control register 1 (INTFOCTL1).
r_intfo_interrupt	Performs processing in response to the external maskable interrupt INTFO.

R_INTC_Create

Performs initialization necessary to control the interrupt functions.

[Syntax]

```
void    R_INTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTC_Create_UserInit

Performs user-defined initialization relating to the interrupt functions.

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

[Syntax]

```
void    R_INTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intc*n*_interrupt

Performs processing in response to the external maskable interrupt INTP*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_intcn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_intcn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Start

Enables the acceptance of the external maskable interrupts INTP*n*.

[Syntax]

```
void    R_INTCn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Stop

Disables the acceptance of the external maskable interrupts INTP*n*.

[Syntax]

```
void    R_INTCn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_intclr*n*_interrupt

Performs processing in response to the external maskable interrupt INTPLR*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTPLR*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_intclrn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_intclrn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR*n*_Start

Enables the acceptance of the external maskable interrupts INTPLR*n*.

[Syntax]

```
void    R_INTCLRn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR*n*_Stop

Disables the acceptance of the external maskable interrupts INTPLR*n*.

[Syntax]

```
void    R_INTCLRn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_intrtcic*n*_interrupt

Performs processing in response to the external maskable interrupt INTRTCIC*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTRTCIC*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_intrtcicn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_intrtcicn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTRTCIC*n*_Start

Enables the acceptance of the external maskable interrupts INTRTCIC*n*.

[Syntax]

```
void    R_INTRTCICn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTRTCIC*n*_Stop

Disables the acceptance of the external maskable interrupts INTR*TCICn*.

[Syntax]

```
void    R_INTRTCICn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTFO_Start

Enables the acceptance of the external maskable interrupts INTFO.

[Syntax]

```
void    R_INTFO_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTFO_Stop

Disables the acceptance of the external maskable interrupts INTFO.

[Syntax]

```
void    R_INTFO_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTFO_ClearFlag

Clears INTFCLR flag of Interrupt flag output control register 1 (INTFOCTL1).

[Syntax]

```
void    R_INTFO_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intfo_interrupt

Performs processing in response to the external maskable interrupt INTFO.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTFO.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_intfo_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_intfo_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.39 Key interrupt function

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

Table 3.39 API Functions: [Key Interrupt Function]

API Function Name	Function
R_KEY_Create	Performs initialization necessary to control the key interrupt function.
R_KEY_Create_UserInit	Performs user-defined initialization relating to the key interrupt function.
r_key_interrupt	Performs processing in response to the key interrupt INTKR.
R_KEY_Start	Enables the acceptance of the key interrupt INTKR.
R_KEY_Stop	Disables the acceptance of the key interrupt INTKR.

R_KEY_Create

Performs initialization necessary to control the key interrupt function.

[Syntax]

```
void    R_KEY_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create_UserInit

Performs user-defined initialization relating to the key interrupt function.

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

[Syntax]

```
void    R_KEY_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_key_interrupt

Performs processing in response to the key interrupt INTKR.

Remark This API function is called as the interrupt process corresponding to the key interrupt INTKR.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_key_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_key_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Start

Enables the acceptance of the key interrupt INTKR.

[Syntax]

```
void    R_KEY_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Stop

Disables the acceptance of the key interrupt INTKR.

[Syntax]

```
void    R_KEY_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.40 Voltage detector

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

Table 3.40 API Functions: [Voltage Detector]

API Function Name	Function
R_LVD_Create	Performs initialization necessary to control the voltage detector.
R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
r_lvd_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
r_lvd_vddinterrupt	Performs processing in response to the voltage detection of VDD pin interrupt INTLVDVDD.
r_lvd_vbatinterrupt	Performs processing in response to the voltage detection of VBAT pin interrupt INTLVDVBAT.
r_lvd_vrtcinterrupt	Performs processing in response to the voltage detection of VRTC pin interrupt INTLVDVRTC
r_lvd_exlvdinterrupt	Performs processing in response to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.
R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).
R_LVD_Start_VDD	Enables operation of VDD pin voltage detection.
R_LVD_Start_VBAT	Enables operation of VBAT pin voltage detection.
R_LVD_Start_VRTC	Enables operation of VRTC pin voltage detection.
R_LVD_Start_EXLVD	Enables operation of EXLVD pin voltage detection.
R_LVD_Stop_VDD	Disables operation of VDD pin voltage detection.
R_LVD_Stop_VBAT	Disables operation of VBAT pin voltage detection.
R_LVD_Stop_VRTC	Disables operation of VRTC pin voltage detection.
R_LVD_Stop_EXLVD	Disables operation of EXLVD pin voltage detection.
R_LVI_Create	Performs initialization necessary to control the voltage detector.
R_LVI_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
r_lvi_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
R_LVI_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

R_LVD_Create

Performs initialization necessary to control the voltage detector.

[Syntax]

```
void    R_LVD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

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

[Syntax]

```
void    R_LVD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvd_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_lvd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vddinterrupt

Performs processing in response to the voltage detection of VDD pin interrupt INTLVDVDD.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VDD pin interrupt INTLVDVDD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvd_vddinterrup ( void );
```

CC-RL Compiler

```
static  void  __near r_lvd_vddinterrup ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vbatinterrupt

Performs processing in response to the voltage detection of VBAT pin interrupt INTLVDVBAT.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VBAT pin interrupt INTLVDVBAT.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvd_vbatinterrup ( void );
```

CC-RL Compiler

```
static  void  __near r_lvd_vbatinterrup ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vrtcinterrupt

Performs processing in response to the voltage detection of VRTC pin interrupt INTLVDVRTC.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VRTC pin interrupt INTLVDVRTC.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvd_vrtcinterrup ( void );
```

CC-RL Compiler

```
static  void  __near r_lvd_vrtcinterrup ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_exlvdinterrupt

Performs processing in response to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.

Remark This API function is called as the interrupt process corresponding to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvd_exlvdinterrup ( void );
```

CC-RL Compiler

```
static  void  __near r_lvd_exlvdinterrup ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_InterruptMode_Start

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

[Syntax]

```
void    R_LVD_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VDD

Enables operation of VDD pin voltage detection.

[Syntax]

```
void    R_LVD_Start_VDD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VBAT

Enables operation of VBAT pin voltage detection.

[Syntax]

```
void    R_LVD_Start_VBAT ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VRTC

Enables operation of VRTC pin voltage detection.

[Syntax]

```
void    R_LVD_Start_VRTC ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_EXLVD

Enables operation of EXLVD pin voltage detection.

[Syntax]

```
void    R_LVD_Start_EXLVD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VDD

Disables operation of VDD pin voltage detection.

[Syntax]

```
void    R_LVD_Stop_VDD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VBAT

Disables operation of VBAT pin voltage detection.

[Syntax]

```
void    R_LVD_Stop_VBAT ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VRTC

Disables operation of VRTC pin voltage detection.

[Syntax]

```
void    R_LVD_Stop_VRTC ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_EXLVD

Disables operation of EXLVD pin voltage detection.

[Syntax]

```
void    R_LVD_Stop_EXLVD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_Create

Performs initialization necessary to control the voltage detector.

[Syntax]

```
void    R_LVI_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

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

[Syntax]

```
void    R_LVI_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvi_interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_lvi_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_lvi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_InterruptMode_Start

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

[Syntax]

```
void    R_LVI_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.41 Battery backup function

Below is a list of API functions output by the Code Generator for battery backup function use.

Table 3.41 API Functions: [Battery Backup Function]

API Function Name	Function
R_BUP_Create	Performs initialization necessary to control the battery backup function.
R_BUP_Create_UserInit	Performs user-defined initialization relating to the battery backup function.
r_bup_interrupt	Performs processing in response to the power switching detection interrupt INTVBAT.
R_BUP_Start	Enables operation of battery backup function.
R_BUP_Stop	Disables operation of battery backup function.

R_BUP_Create

Performs initialization necessary to control the battery backup function.

[Syntax]

```
void    R_BUP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Create_UserInit

Performs user-defined initialization relating to the battery backup function.

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

[Syntax]

```
void    R_BUP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bup_interrupt

Performs processing in response to the power switching detection interrupt INTVBAT.

Remark This API function is called as the interrupt process corresponding to the power switching detection interrupt INTVBAT.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_bup_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_bup_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Start

Enables operation of battery backup function.

[Syntax]

```
void    R_BUP_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Stop

Disables operation of battery backup function.

[Syntax]

```
void    R_BUP_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.42 Oscillation stop detector

Below is a list of API functions output by the Code Generator for oscillation stop detector use.

Table 3.42 API Functions: [Oscillation Stop Detector]

API Function Name	Function
R_OSDC_Create	Performs initialization necessary to control the oscillation stop detector.
R_OSDC_Create_UserInit	Performs user-defined initialization relating to the oscillation stop detector.
r_osdc_interrupt	Performs processing in response to the oscillation stop detection interrupt INTOSDC.
R_OSDC_Start	Enables operation of oscillation stop detector.
R_OSDC_Stop	Disables operation of oscillation stop detector.
R_OSDC_Set_PowerOff	Halts the clock supplied to the oscillation stop detector.
R_OSDC_Reset	Reset the oscillation stop detector.

R_OSDC_Create

Performs initialization necessary to control the oscillation stop detector.

[Syntax]

```
void    R_OSDC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Create_UserInit

Performs user-defined initialization relating to the oscillation stop detector.

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

[Syntax]

```
void    R_OSDC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_osdc_interrupt

Performs processing in response to the oscillation stop detection interrupt INTOSDC.

Remark This API function is called as the interrupt process corresponding to the oscillation stop detection interrupt INTOSDC.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_osdc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_osdc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Start

Enables operation of oscillation stop detector.

[Syntax]

```
void    R_OSDC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Stop

Disables operation of oscillation stop detector.

[Syntax]

```
void    R_OSDC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Set_PowerOff

Halts the clock supplied to the oscillation stop detector.

Remark Calling this API function changes the oscillation stop detector to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_OSDC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Reset

Reset the oscillation stop detector.

[Syntax]

```
void    R_OSDC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.43 SPI interface

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

Table 3.43 API Functions: [SPI Interface]

API Function Name	Function
R_SAIC_Create	Performs initialization necessary to control the SPI interface.
R_SAIC_Create_UserInit	Performs user-defined initialization relating to the SPI interface.
R_SAIC_Write	Starts SPI data transmission.
R_SAIC_Read	Starts SPI data reception.
R_SPI_Create	Performs initialization necessary to control the SPI interface.
R_SPI_Create_UserInit	Performs user-defined initialization relating to the SPI interface.
R_SPI_Write	Starts SPI data transmission.
R_SPI_Read	Starts SPI data reception.

R_SAIC_Create

Performs initialization necessary to control the SPI interface.

[Syntax]

```
void    R_SAIC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SAIC_Create_UserInit

Performs user-defined initialization relating to the SPI interface.

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

[Syntax]

```
void    R_SAIC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SAIC_Write

Starts SPI data transmission.

[Syntax]

```
void    R_SAIC_Write ( const smartanalog_t * p_saic_data );
```

[Argument(s)]

I/O	Argument	Description
I	`const smartanalog_t * p_saic_data;`	Pointer to area storing the transmission data

[Return value]

None.

R_SAIC_Read

Starts SPI data reception.

[Syntax]

```
void    R_SAIC_Read ( const smartanalog_t * p_saic_data, smartanalog_t *
p_saic_read_buf );
```

[Argument(s)]

I/O	Argument	Description
O	`const smartanalog_t * p_saic_data;`	Pointer to area to store the received data
O	`smartanalog_t * p_saic_read_buf;`	Pointer to a buffer to store the received data

[Return value]

None.

R_SPI_Create

Performs initialization necessary to control the SPI interface.

[Syntax]

```
void    R_SPI_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPI_Create_UserInit

Performs user-defined initialization relating to the SPI interface.

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

[Syntax]

```
void    R_SPI_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPI_Write

Starts SPI data transmission.

[Syntax]

```
void    R_SPI_Write ( const smartanalog_t * p_saic_data );
```

[Argument(s)]

I/O	Argument	Description
I	`const smartanalog_t * p_saic_data;`	Pointer to area storing the transmission data

[Return value]

None.

R_SPI_Read

Starts SPI data reception.

[Syntax]

```
void    R_SPI_Read ( const smartanalog_t * p_saic_data, smartanalog_t * p_saic_read_buf );
```

[Argument(s)]

I/O	Argument	Description
O	`const smartanalog_t * p_saic_data;`	Pointer to area to store the received data
O	`smartanalog_t * p_saic_read_buf;`	Pointer to a buffer to store the received data

[Return value]

None.

3.2.44 Operational amplifier

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

Table 3.44 API Functions: [Operational amplifier]

API Function Name	Function
R_OPAMP_Create	Performs initialization necessary to control the operational amplifier.
R_OPAMP_Create_UserInit	Performs user-defined initialization relatede to the operational amplifier.
R_OPAMP_Set_ReferenceCurcuitOn	Enables operational amplifier reference current circuit.
R_OPAMP_Set_ReferenceCircuitOff	Disables operational amplifier reference current circuit.
R_OPAMPn_Start	Starts operational amplifier of unit *n*.
R_OPAMPn_Stop	Stops operational amplifier of unit *n*.
R_OPAMPn_Set_PrechargeOn	Starts precharging of the external capacitor of the operational amplifier *n*.
R_OPAMPn_Set_PrechargeOff	Performs user-defined initialization relatede to the operational amplifier.

R_OPAMP_Create

Performs initialization necessary to control the operational amplifier.

[Syntax]

```
void    R_OPAMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Create_UserInit

Performs user-defined initialization relating to the operational amplifier.

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

[Syntax]

```
void    R_OPAMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Set_ReferenceCurcuitOn

Enables operational amplifier reference current circuit.

[Syntax]

```
void    R_OPAMP_Set_ReferenceCircuitOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Set_ReferenceCircuitOff

Disables operational amplifier reference current circuit.

[Syntax]

```
void    R_OPAMP_Set_ReferenceCircuitOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Start

Starts operational amplifier of unit *n*.

[Syntax]

```
void    R_OPAMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Stop

Stops operational amplifier of unit *n*.

[Syntax]

```
void    R_OPAMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Set_PrechargeOn

Starts precharging of the external capacitor of the operational amplifier *n*.

[Syntax]

```
void    R_OPAMPn_Set_PrechargeOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Set_PrechargeOff

Stops precharging of the external capacitor of the operational amplifier *n*.

[Syntax]

```
void    R_OPAMPn_Set_PrechargeOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.45 Data operation circuit

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

Table 3.45 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 related to the data operation circuit.
r_doc_interrupt	Performs processing in response to the DOC operation result detection interrupt INTDOC.
R_DOC_SetMode	Configures the operation mode of data operation circuit.
R_DOC_WriteData	Writes new data to compare, add or subtract.
R_DOC_GetResult	Gets result of addition or subtraction.
R_DOC_ClearFlag	Clears DOPCF flag of DOC control register (DOCR).
R_DOC_Set_PowerOff	Stops the clock supplied for data operation circuit.
R_DOC_Reset	Resets Data opration circuit module.

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_interrupt

Performs processing in response to the data operation circuit interrupt INTDOC.

Remark This API function is called as the interrupt process corresponding to the data operation circuit interrupt INTDOC.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_doc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_doc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

Configures the operation mode of data operation circuit.

[Syntax]

```
#include     “r_cg_macrodriver.h”
#include     “r_cg_doc.h”
MD_STATUS     R_DOC_SetMode ( doc_mode_t mode, uint16_t value);
```

[Argument(s)]

I/O	Argument	Description
I	doc_mode_t *mode*;	Operation mode of data operation circuit ADDTION: Data addition mode SUBTRACTION: Data subtraction mode COMPARE_MATCH: Data comparison mode (Detection Condition: Data match is detected) COMPARE_MISMATCH: Data subtraction mode (Detection Condition: Data mismatch is detected)
I	uint16_t *value*;	Data addition and data subtraction : Results of operations Data comparison : 16-bit data for use as a reference

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_DOC_WriteData

Writes new data to compare, add or subtract.

Remark Write data to DODIR register.

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
O	uint16_t *data*;	data to compare, add or subtract

[Return value]

None.

R_DOC_GetResult

Gets result of addition or subtraction.

[Syntax]

```
#include     “r_cg_macrodriver.h”
void          R_DOC_GetResult ( uint16_t *const data );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t *const data	pointer to where result will be stored

[Return value]

None.

R_DOC_ClearFlag

Clears DOPCF flag of DOC control register (DOCR).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DOC_Set_PowerOff

Stops the clock supplied for data operation circuit.

[Syntax]

```
void    R_DOC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Reset

Resets Data operation circuit module.

[Syntax]

```
void    R_DOC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.46 32-bit Multiply-accumulator

Below is a list of API functions output by the Code Generator for 32-bit Multiply-accumulator use.

Table 3.46 API Functions: [32-bit Multiply-accumulator]

API Function Name	Function
R_MAC32Bit_Create	Performs initialization necessary to control the data operation circuit.
R_MAC32Bit_Create_UserInit	Performs user-defined initialization related to the data operation circuit.
r_mac32bit_interrupt_flow	Performs processing in response to the DOC operation result detection interrupt INTDOC.
R_MAC32Bit_Reset	Gets result of addition or subtraction.
R_MAC32Bit_Set_PowerOff	Clears DOPCF flag of DOC control register (DOCR).
R_MAC32bit_MULUnsigned	Operates the unsigned multiply.
R_MAC32Bit_MULSigned	Operates the signed multiply.
R_MAC32Bit_MACUnsigned	Operates the unsigned multiply-accumulate.
R_MAC32Bit_MACSigned	Operates the signed multiply-accumulate.

R_MAC32Bit_Create

Performs initialization necessary to control the 32-bit Multiply-accumulator.

[Syntax]

```
void    R_MAC32Bit_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Create_UserInit

Performs user-defined initialization relating to the 32-bit Multiply-accumulator.

Remark This API function is called as the R_MAC32Bit_Create callback routine.

[Syntax]

```
void    R_MAC32Bit_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mac32bit_interrupt_flow

Performs processing in response to the 32-bit Multiply-accumulator interrupt INTMACLOF.

Remark This API function is called as the interrupt process corresponding to the 32-bit Multiply-accumulator interrupt INTMACLOF.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_mac32bit_interrupt_flow ( void );
```

CC-RL Compiler

```
static  void  __near r_mac32bit_interrupt_flow ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Reset

Resets the 32-bit Multiply-accumulator.

[Syntax]

```
void    R_MAC32Bit_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Set_PowerOff

Stops the cock supplied for the 32-bit Multiply-accumulator.

[Syntax]

```
void    R_MAC32Bit_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32bit_MULUnsigned

Operates the unsigned multiply.

[Syntax]

```
#include     “r_cg_macrodriver.h”
#include    “r_cg_mac32bit.h”
void    R_MAC32Bit_MULUnsigned(uint32_t data_a, uint32_t data_b, mac32bit_uint64_t *
buffer_64bit);
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t data_a`	Multiplicand
I	`uint32_t data_b`	Multiplier
O	`mac32bit_uint64_t * buffer_64bit`	Multiplier result

Remark Below is an example of the structure mac32bit_uint64_t for the multiplier result.

```
typedef struct
{
          uint16_t low_low;
          uint16_t low_high;
          uint16_t high_low;
          uint16_t high_high;
} mac32bit_uint64_t;
```

[Return value]

None.

R_MAC32Bit_MULSigned

Operates the signed multiply.

[Syntax]

```
#include     “r_cg_macrodriver.h”
#include    “r_cg_mac32bit.h”
void    R_MAC32Bit_MULSigned(int32_t data_a, int32_t data_b, mac32bit_int64_t *
buffer_64bit);
```

[Argument(s)]

I/O	Argument	Description
I	`int32_t data_a`	Multiplicand
I	`int32_t data_b`	Multiplier
O	`mac32bit_int64_t * buffer_64bit`	Multiplier result

Remark Below is an example of the structure mac32bit_int64_t for the multiplier result.

```
typedef struct
{
          int16_t low_low;
          int16_t low_high;
          int16_t high_low;
          int16_t high_high;
} mac32bit_int64_t;
```

[Return value]

None.

R_MAC32Bit_MACUnsigned

Operates the unsigned multiply-accumulate.

[Syntax]

```
#include     “r_cg_macrodriver.h”
#include    “r_cg_mac32bit.h”
void    R_MAC32Bit_MACUnsigned(uint32_t data_a, uint32_t data_b, mac32bit_uint64_t *
buffer_64bit);
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t data_a`	Multiplicand
I	`uint32_t data_b`	Multiplier
O	`mac32bit_uint64_t * buffer_64bit`	Accumulation initial value ／ Multiplier result

Remark See R_MAC32bit_MULUnsigned for details about the `mac32bit_uint64_t`.

[Return value]

None.

R_MAC32Bit_MACSigned

Operates the signed multiply-accumulate.

[Syntax]

```
#include     “r_cg_macrodriver.h”
#include    “r_cg_mac32bit.h”
void    R_MAC32Bit_MACSigned(int32_t data_a, int32_t data_b, mac32bit_int64_t *
buffer_64bit);
```

[Argument(s)]

I/O	Argument	Description
I	`int32_t data_a`	Multiplicand
I	`int32_t data_b`	Multiplier
O	`mac32bit_int64_t * buffer_64bit`	Accumulation initial value ／ Multiplier result

Remark See R_MAC32Bit_MULSigned for details about the `mac32bit_int64_t`.

[Return value]

None.

3.2.47 12-bit A/D converter

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

Table 3.47 API Functions: [12-bit A/D Converter]

API Function Name	Function
R_12ADC_Create	Performs initialization necessary to control the 12-bit A/D converter.
R_12ADC_Create_UserInit	Performs user-defined initialization relating to the 12-bit A/D converter.
r_12adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
R_12ADC_Start	Starts A/D conversion.
R_12ADC_Stop	Ends A/D conversion.
R_12ADC_Get_ValueResult	Reads the results of A/D conversion (12 bits).
R_12ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
R_12ADC_TemperatureSensorOutput_On	Enables 12-bit A/D converter temperature sensor output circuit.
R_12ADC_TemperatureSensorOutput_Off	Disables12-bit A/D converter temperature sensor output circuit.
R_12ADC_InternalReferenceVoltage_On	Enables 12-bit A/D converter reference voltage circuit.
R_12ADC_InternalReferenceVoltage_Off	Disables12-bit A/D converter reference voltage circuit.
R_12ADC_Set_PowerOff	Halts the clock supplied to the 12-bit A/D converter.

R_12ADC_Create

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

[Syntax]

```
void    R_12ADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Create_UserInit

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

Remark This API function is called as the R_12ADC_Create callback routine.

[Syntax]

```
void    R_12ADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_12adc_interrupt

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

Remark This API function is called as the interrupt process corresponding to the A/D conversion end interrupt INTAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_12adc_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_12adc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to R_12ADC_Create and the call to this API function.

[Syntax]

```
void    R_12ADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call R_12ADC_Set_PowerOff after the process of this API function completes.

[Syntax]

```
void    R_12ADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Get_ValueResult

Reads the results of A/D conversion (12 bits).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_12ADC_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
I	`ad_channel_t channel`	Analog voltage input pin
O	`uint16_t  * const buffer;`	Pointer to area in which to store read results of A/D conversion

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_12ADC_Set_ADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in argument channel is set to A/D channel select register A0 (ADANSA0) or A/D conversion extended input control register (ADEXICR).

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_12adc.h"
MD_STATUS   R_12ADC_Set_ADChannel ( ad_sel_register_t register, uint16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	`ad_sel_register_t register;`	Set to selected register SEL_ADANSA0: A/D channel select register A0 (ADANSA0) SEL_ADEXICR: A/D conversion extended input control register (ADEXICR)
I	`uint16_t data;`	Set to selected register value

Remark See the header file r_cg_12adc.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_12ADC_TemperatureSensorOutput_On

Enables 12-bit A/D converter temperature sensor output circuit.

[Syntax]

```
void    R_12ADC_TemperatureSensorOutput_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_TemperatureSensorOutput_Off

Disables 12-bit A/D converter temperature sensor output circuit.

[Syntax]

```
void    R_12ADC_TemperatureSensorOutput_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_InternalReferenceVoltage_On

Enables 12-bit A/D converter reference voltage circuit.

[Syntax]

```
void    R_12ADC_InternalReferenceVoltage_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_InternalReferenceVoltage_Off

Disables 12-bit A/D converter reference voltage circuit.

[Syntax]

```
void    R_12ADC_InternalReferenceVoltage_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Set_PowerOff

Halts the clock supplied to the 12-bit A/D converter.

Remark Calling this API function changes the A/D converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_12ADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.48 12-bit D/A converter

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

Table 3.48 API Functions: [12-bit D/A Converter]

API Function Name	Function
R_12DA_Create	Performs initialization necessary to control the 12-bit D/A converter.
R_12DA_Create_UserInit	Performs user-defined initialization relating to the 12-bit D/A converter.
R_12DAn_Start	Starts D/A conversion.
R_12DAn_Stop	Ends D/A conversion.
R_12DA_Set_PowerOff	Halts the clock supplied to the 12-bit D/A converter.
R_12DAn_Set_ConversionValue	Sets the analog voltage output to the ANO*n* pin.

R_12DA_Create

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

[Syntax]

```
void    R_12DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA_Create_UserInit

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

Remark This API function is called as the R_12DA_Create callback routine.

[Syntax]

```
void    R_12DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Start

Starts D/A conversion.

[Syntax]

```
void    R_12DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Stop

Ends D/A conversion.

[Syntax]

```
void    R_12DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_12DA_Set_PowerOff

Halts the clock supplied to the 12-bit D/A converter.

Remark Calling this API function changes the 12-bit D/A converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_12DA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Set_ConversionValue

Sets the analog voltage output to the ANO*n* pin.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_12DAn_Set_ConversionValue ( uint16_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t reg_value;`	D/A conversion value.

[Return value]

None.

3.2.49 Operational amplifier and Analog switch

Below is a list of API functions output by the Code Generator for Operational Amplifier and Analog switch use.

Table 3.49 API Functions: [Operational amplifier and Analog switch]

API Function Name	Function
R_AMPANSW_Create	Performs initialization necessary to control the Operational amplifier and Analog switch.
R_AMPANSW_Create_UserInit	Performs user-defined initialization relating to the Operational amplifier and Analog switch.
R_OPAMPm_Set_ReferenceCurcuitOn	Enables operational amplifier reference current circuit.
R_OPAMPm_Set_ReferenceCircuitOff	Disables operational amplifier reference current circuit.
R_OPAMPm_Start	Starts operational amplifier of unit *m*.
R_OPAMPm_Stop	Stops operational amplifier of unit *m*.
R_ANSW_ChargePumpm_On	Enables analog switch of unit *m*.
R_ANSW_ChargePumpm_Off	Disables analog switch of unit *m*.

R_AMPANSW_Create

Performs initialization necessary to control the Operational amplifier and Analog switch.

[Syntax]

```
void    R_AMPANSW_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMPANSW_Create_UserInit

Performs user-defined initialization relating to the Operational amplifier and Analog switch.

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

[Syntax]

```
void    R_AMPANSW_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Set_ReferenceCurcuitOn

Enables operational amplifier reference current circuit.

[Syntax]

```
void    R_OPAMPm_Set_ReferenceCircuitOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Set_ReferenceCircuitOff

Disables operational amplifier reference current circuit.

[Syntax]

```
void    R_OPAMPm_Set_ReferenceCircuitOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Start

Starts operational amplifier of unit *m*.

[Syntax]

```
void    R_OPAMPm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Stop

Stops operational amplifier of unit *m*.

[Syntax]

```
void    R_OPAMPm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ANSW_ChargePump*m*_On

Enables analog switch of unit *m*.

[Syntax]

```
void    R_ANSW_ChargePumpm_On ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ANSW_ChargePump*m*_Off

Disables analog switch of unit *m*.

[Syntax]

```
void    R_ANSW_ChargePumpm_Off ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.50 Voltage reference

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

Table 3.50 API Functions: [Voltage Reference]

API Function Name	Function
R_VR_Create	Performs initialization necessary to control the Voltage reference.
R_VR_Create_UserInit	Performs user-defined initialization relating to the Voltage reference.
R_VR_Start	Enables operation of Voltage reference.
R_VR_Stop	Disables operation of Voltage reference.

R_VR_Create

Performs initialization necessary to control the Voltage reference.

[Syntax]

```
void    R_VR_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Create_UserInit

Performs user-defined initialization relating to the Voltage referdetector.

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

[Syntax]

```
void    R_VR_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Start

Enables operation of Voltage reference.

[Syntax]

```
void    R_VR_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Stop

Disables operation of Voltage reference.

[Syntax]

```
void    R_VR_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.51 Sampling output timer detector

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

Table 3.51 API Functions: [Sampling output timer detector]

API Function Name	Function
R_SMOTD_Create	Performs initialization necessary to control the Sampling output timer detector.
R_SMOTD_Create_UserInit	Performs user-defined initialization relating to the Sampling output timer detector.
r_smotd_counterA_interrupt	Performs processing in response to the Sampling output timer interval interrupt INTSMOTA.
r_smotd_counterB_interrupt	performs processing in response to the Sampling output timer compare match interrupt INTSMOTB.
r_smotd_smpn_interrupt	Performs processing int response to the Sampling detector detection interrupt INTSMPn.
R_SMOTD_Start	Starts Sampling output timer detector.
R_SMOTD_Stop	Ends Sampling output timer detector.
R_SMOTD_Set_PowerOff	Halts the clock supplied to the Sampling output timer detector.

R_SMOTD_Create

Performs initialization necessary to control the Sampling output timer detector.

[Syntax]

```
void    R_SMOTD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Create_UserInit

Performs user-defined initialization relating to the Sampling output timer detector.

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

[Syntax]

```
void    R_SMOTD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_counterA_interrupt

Performs processing in response to the Sampling output timer interval interrupt INTSMOTA.

Remark This API function is called as the interrupt process corresponding to the Sampling output timer interval interrupt INTSMOTA.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_smotd_counterA_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_smotd_counterA_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_counterB_interrupt

Performs processing in response to the Sampling output timer compare match interrupt INTSMOTB.

Remark This API function is called as the interrupt process corresponding to the Sampling output timer compare match INTSMOTB.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_smotd_counterB_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_smotd_counterB_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_smp*n*_interrupt

Performs processing in response to the Sampling detector detection interrupt INTSMP*n*.

Remark This API function is called as the interrupt process corresponding to the Sampling detector detection interrupt INTSMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_smotd_smpn_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_smotd_smpn_interrupt ( void );
```

Remark *n* is the sampling input number.

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Start

Starts Sampling output timer detector.

[Syntax]

```
void    R_SMOTD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Stop

Ends Sampling output timer detector.

[Syntax]

```
void    R_SMOTD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Set_PowerOff

Halts the clock supplied to the Sampling output timer detector.

[Syntax]

```
void    R_SMOTD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.52 External signal sampler

Below is a list of API functions output by the Code Generator for External signal sampler use.

Table 3.52 API Functions: [External signal sampler]

API Function Name	Function
R_EXSD_Create	Performs initialization necessary to control the External signal sampler.
R_EXSD_Create_UserInit	Performs user-defined initialization relating to the External signal sampler.
r_exsd_interrupt	Performs processing in response to the External signal sampler edge detection interrupt INTEXSD.
R_EXSD_Start	Starts External signal sampling.
R_EXSD_Stop	Ends External signal sampling.
R_EXSD_Set_PowerOff	Halts the clock supplied to the External signal sampler.

R_EXSD_Create

Performs initialization necessary to control the External signal sampler.

[Syntax]

```
void    R_EXSD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Create_UserInit

Performs user-defined initialization relating to the External signal sampler.

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

[Syntax]

```
void    R_EXSD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_exsd_interrupt

Performs processing in response to the External signal sampler edge detection interrupt INTEXSD.

Remark This API function is called as the interrupt process corresponding to the External signal sampler edge detection interrupt INTEXSD.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_exsd_interrupt ( void );
```

CC-RL Compiler

```
static  void  __near r_exsd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Start

Starts A/D External signal sanpling.

[Syntax]

```
void    R_EXSD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Stop

Ends External signal sampling.

[Syntax]

```
void    R_EXSD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Set_PowerOff

Halts the clock supplied to the External signal sampler.

Remark Calling this API function changes the External signal sampler to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_EXSD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.53 Serial interface UARTMG

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

Table 3.53 API Functions: [Serial interface UARTMG]

API Function Name	Function
R_UARTMGn_Create	Performs initialization necessary to control the serial interface UARTMG.
R_UARTMGn_Create_UserInit	Performs user-defined initialization related to the serial interface UARTMG.
r_uartmgn_interrupt_send	Performs processing in response to the UARTMG transmission completion interrupt INTSTMG*n*.
r_uartmgn_interrupt_receive	Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.
r_uartmgn_interrupt_error	Perfomrs processing in response to the UARTMG reception error interrupt INTSREMG*n*.
R_UARTMGn_Start	Sets UARTMG communication to standby mode..
R_UARTMGn_Stop	Ends UARTMG communication.
R_UARTMGn_Set_PowerOff	Halts the clock supplied to the serial interface UARTMG.
R_UARTMGn_Send	Starts UARTMG data transmission.
R_UARTMGn_Receive	Starts UARTMG data reception.
r_uartmgn_callback_sendend	Performs processing in response to the UARTMG transmission completion interrupt INTSTMG*n*.
r_uartn_callback_receiveend	Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.
r_uartmgn_callback_error	Performs processing in response to the UARTMG reception error interrupt INTSREMG*n*.
r_uartmgn_callback_softwareoverrun	Performs processing in response to detection of overrun error.

R_UARTMG*n*_Create

Performs initialization necessary to control the serial interface UARTMG.

[Syntax]

```
void    R_UARTMGn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Create_UserInit

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

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

[Syntax]

```
void    R_UARTMGn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_send

Performs processing in response to the UARTMG transmission completion interrupt INTSTMGn.

Remark This API function is called as the interrupt process corresponding to the UARTMG transmission completion interrupt INTSTMG*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartmgn_interrupt_send ( void );
```

CC-RL Compiler

```
static  void  __near r_uartmgn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_receive

Performs processing in response to the UARTMG reception completion interrupt INTSRMGn.

Remark This API function is called as the interrupt process corresponding to the UARMG reception completion interrupt INTSR*MGn*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartmgn_interrupt_receive ( void );
```

CC-RL Compiler

```
static  void  __near r_uartmgn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_error

Performs processing in response to the UARTMG reception error interrupt INTSREMGn.

Remark This API function is called as the interrupt process corresponding to the UARTMG reception error interrupt INTSRE*MGn*.

[Syntax]

CA78K0R Compiler

```
__interrupt static  void    r_uartmgn_interrupt_error ( void );
```

CC-RL Compiler

```
static  void  __near r_uartmgn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Start

Sets UARTMG communication to standby mode.

[Syntax]

```
void    R_UARTMGn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Stop

Ends UARTMG communication.

[Syntax]

```
void    R_UARTMGn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Set_PowerOff

Halts the clock supplied to the serial interface UARTMG.

Remark Calling this API function changes the serial interface UARTMG to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_UARTMGn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Send

Starts UARTMG data transmission.

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

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTMGn_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 specification

R_UARTMG*n*_Receive

Starts UARTMG data reception.

Remarks 1. This API function performs byte-level UART 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. Actual UART reception starts after this API function is called, and R_UARTMGn_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTMGn_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 received data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_uartmg*n*_callback_sendend

Performs processing in response to the UARTMG transmission complition interrupt INTSTMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_send corresponding to the UARTMG transmission completion interrupt INTSTMG*n* (performed when number of transmission data specified by R_UARTMGn_Send argument *tx_num* has been completed).

[Syntax]

```
static  void    r_uartmgn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_callback_receiveend

Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_receive corresponding to the UARTMG reception completion interrupt INTSRMG*n* (performed when number of received data specified by R_UARTMGn_Receive argument *rx_num* has been completed).

[Syntax]

```
static  void    r_uartmgn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_callback_error

Performs processing in response to the UARTMG reception error interrupt INTSREMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_error corresponding to the UARTMG reception error interrupt INTSRE*MGn*.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_uartn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for UART reception error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing error

[Return value]

None.

r_uartmg*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_receive corresponding to the UARTMG reception end interrupt INTSRMG*n* (process performed when the amount of data received is greater than the argument *rx_num* specified for R_UARTMGn_Receive).

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_uartmgn_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint16_t  rx_data;`	Receive data (greater than the argument *rx_num* specified for R_UARTMGn_Receive)

[Return value]

None.

3.2.54 Amplifier unit

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

Table 3.54 API Functions: [Amplifier unit]

API Function Name	Function
R_AMP_Create	Performs initialization necessary to control the amplifier unit.
R_AMP_Create_UserInit	Performs user-defined initialization relatede to the amplifie unit.
R_AMP_Set_PowerOn	Enables amplifier unit power supply.
R_AMP_Set_PowerOff	Disables amplifier unit power supply.
R_PGA1_Start	Starts instrumentation amplifier 1.
R_OPAMPn_Stop	Stops instrumentation amplifier 1.
R_AMPn_Start	Starts operational amplifier *n*.
R_AMPn_Stop	Stops operational amplifier *n*.

R_AMP_Create

Performs initialization necessary to control the amplifier unit.

[Syntax]

```
void    R_AMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Create_UserInit

Performs user-defined initialization relating to the amplifier unit.

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

[Syntax]

```
void    R_AMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Set_PowerOn

Enables amplifier unit power supply.

[Syntax]

```
void    R_AMP_Set_PowerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Set_PowerOff

Disables amplifier unit power supply.

[Syntax]

```
void    R_AMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA1_Start

Starts instrumentation amplifier 1.

[Syntax]

```
void    R_PGA1_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Stop

Stops instrumentation amplifier 1.

[Syntax]

```
void    R_PGA1_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP*n*_Start

Starts operational amplifier *n*.

[Syntax]

```
void    R_AMPn_Start ( void );
```

Remark *n* is the operational amplifier unit number.

[Argument(s)]

None.

[Return value]

None.

R_AMP*n*_Stop

Stops operational amplifier *n*.

[Syntax]

```
void    R_AMPn_Stop ( void );
```

Remark *n* is the operational amplifier unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.55 Data flash libraries

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

Table 3.55 API Functions: [Data Flash Libraries]

API Function Name	Function
R_FDL_Create	Performs initialization necessary to control the Data Flash Libraries.
R_FDL_Open	Starts the Data Flash Libraries.
R_FDL_Close	Stop the Data Flash Libraries.
R_FDL_Write	Writes the data to Data Flash Memories.
R_FDL_Read	Reads the data from Data Flash Memories.
R_FDL_Erase	Erases data for Data Flash Memories.

R_FDL_Create

Performs initialization necessary to control the Data Flash Libraries.

[Syntax]

```
void    R_FDL_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FDL_Open

Starts the Data Flash Libraries.

[Syntax]

```
void    R_FDL_Open ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FDL_Close

Stops the Data Flash Libraries.

[Syntax]

```
void    R_SAUm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_FDL_Write

Writes the data to Data Flash Memories.

[Syntax]

```
pfdl_status_t R_FDL_Write ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecounter);
```

Remark *m* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	`pfld_u16 index;`	Writing start address of Data Flash Memories
I	`pfdl_u08 * buffer;`	Pointer to a buffer to store the write data
I	`pfdl_u16 bytecounter;`	Total amount of data to wire

[Return value]

Macro	Description
PFDL_OK	Normal completion
PFDL_BUSY	During execution of the other commands.
PFDL_ERR_WRITE	Error of the writing
PFDL_ERR_PARAMETER	Error of the parameters

R_FDL_Read

Reads the data from Data Flash Memories.

[Syntax]

```
pfdl_status_t R_FDL_Read ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecounter );
```

[Argument(s)]

I/O	Argument	Description
I	`pfdl_u16 index;`	Reading start address of the Data Flash Memories
O	`pfdl_u08 * buffer;`	Pointer to a buffer to store the read data
I	`uint16_t  tx_num;`	Total amount of data to read

[Return value]

Macro	Description
PFDL_OK	Normal completion
PFDL_BUSY	During execution of the other commands.
PFDL_ERR_PARAMETER	Error of the parameters

R_FDL_Erase

Erases the block of Data Flash Memories.

[Syntax]

```
void    R_FDL_Erase ( pfdl_u16 blockno );
```

[Argument(s)]

I/O	Argument	Description
I	`pfdl_u16 blockno;`	Erase no block for Data Flash Memories

[Return value]

Macro	Description
PFDL_OK	Normal completion
PFDL_ERR_ERASE	Error of the erasing
PFDL_ERR_PARAMETER	Error of the parameters

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug 01, 2015		First Edition issued
1.01	Dec 01, 2014		2. OUTPUT FILES TimerRJ, TimerRD API addition
			2. OUTPUT FILES Comparator/Programmable gain amplifier API addition
			3.2.5 Timer RJ API addition
			3.2.6 Timer RD API addition
			3.2.24 Comparator/Programmable gain amplifier chapter addition
1.02	Aug 01, 2015		2. OUTPUT FILES API addition ・Timer RX ・PGA+DS A/D converter ・Configurable Amp ・D/A converter ・Voltage detector
		98 - 105	3.2.8 Timer RX chapter addition
		208 - 218	3.2.19 PGA+DS A/D converter chapter addition
		235 - 240	3.2.21 Configurable Amp chapter addition
		265 - 266	3.2.24 D/A converter API addition
		439 - 442	3.2.39 Voltage detector API addition
1.03	Mar 01, 2016	7,11, 13,16, 17,18, 19	2. OUTPUT FILES API addition ・High-speed on-chip Oscillator clock Frequency Correction function ・Real-time clock ・Temperature sensor ・24-bit DS A/D converter ・Serial interface IICA ・Interrupt functions ・Voltage detector ・Oscillation stop detector ・32-bit Multiply-accumulator
		44 — 49	3.2.4 High-speed on-chip Oscillator clock Frequency Correction function chapter addition
		166 — 190	3.2.13 Real-time clock API addition
		271	3.2.23 Temperature sensor API addition

Rev.	Date	Description	
		Page	Summary
		282	3.2.24 24-bit DS A/D converter API addition
		323	3.2.32 Serial interface IICA API addition
		457 — 459	3.2.38 Interrupt functions API addition
		470 — 482	3.2.40 Voltage detector API addition
		500	3.2.42 Oscillation stop detector API addition
		525 — 534	3.2.46 32-bit Multiply-accumulator chapter addition
1.04	2016.10.01	8, 9, 11, 13, 14, 16, 17	2. OUTPUT FILES API addition ・Timer array unit(R_TAUmReset) ・16-bit timer KB(R_TMR_KB*m*_ForcedOutput_*mn*_Start, R_TMR_KB*m*_ForcedOutput_*mn*_Stop, R_TMR_KB*m*_Reset) ・A/D converter(R_ADC_Reset) ・D/A converter(R_DAC*n*_Create, R_DAC_Reset) ・Programmable gain amplifier(R_PGA_Reset) ・LCD controller/driver(R_LCD_VoltageOn, R_LCD_VoltageOff) ・Interrupt functions(R_INTFO_Start, R_INTFO_Stop, R_INTFO_ClearFlag, r_intfo_interrupt)
		9, 17	2. OUTPUT FILES File addition ・16-bit timer KB(r_cg_tmkb.c, r_cg_tmb_user.c, r_cg_tmkb.h) ・Key interrupt function(r_cg_key.c, r_cg_key_user.c, r_cg_key.h)
		19, 20	2. OUTPUT FILES Function additional ・12-bit A/D converter ・12-bit D/A converter ・Operational amplifier and Analog switch ・Voltage reference
		22	Table 3.1.) API Functions: [Common] Note update ・hdwinit
		23, 24	3.2.1. Common Note update ・hdwinit ・R_Systeminit
		115, 125-127	3.2.10.16-bit timer KB API addition ・R_TMR_KB*m*_ForcedOutput_*mn*_Start ・R_TMR_KB*m*_ForcedOutput_*mn*_Stop ・R_TMR_KB*m*_Reset
		286, 295-296	3.2.25.D/A converter API addition ・R_DAC*n*_Create ・R_DAC_Reset

Rev.	Date	Description	
		Page	Summary
		295, 300	3.2.26.Programmable gain amplifier APIaddition ・R_PGA_Reset
		414, 423-424	3.2.33.LCD controller/driver API addition ・R_LCD_VoltageOn ・R_LCD_VoltageOff
		448, 460-463	3.2.38.Interrupt functions API addition ・R_INTFO_Start ・R_INTFO_Stop ・R_INTFO_ClearFlag ・r_intfo_interrupt
		536-547	3.2.47.12-bit A/D converter chapter addition API addition ・R_12ADC_Create ・R_12ADC_Create_UserInit ・r_12adc_interrupt ・R_12ADC_Start ・R_12ADC_Stop ・R_12ADC_Get_ValueResult ・R_12ADC_Set_ADChannel ・R_12ADC_TemperatureSensorOutput_On ・R_12ADC_TemperatureSensorOutput_Off ・R_12ADC_InternalReferenceVoltage_On ・R_12ADC_InternalReferenceVoltage_Off ・R_12ADC_Set_PowerOff
		551-557	3.2.48.12-bit D/A converter chapter addition API addition ・R_12DA_Create ・R_12DA_Create_UserInit ・R_12DA*n*_Start ・R_12DA*n*_Stop ・R_12DA*n*_Set_ConversionValue ・R_12DA_Set_PowerOff
		561-569	3.2.49.Operational amplifier and Analog switch chapter addition API addition ・R_AMPANSW_Create ・R_AMPANSW_Create_UserInit ・R_OPAMP*m*_Set_ReferenceCircuitOn ・R_OPAMP*m*_Set_ReferenceCircuitOff ・R_OPAMP*m*_Start ・R_OPAMP*m*_Stop ・R_ANSW_ChargePump*m*_On ・R_ANSW_ChargePump*m*_Off

Rev.	Date	Description	
		Page	Summary
		571-575	3.2.50.Voltage reference chapter addition API addition ・R_VR_Create ・R_VR_Create_UserInit ・R_VR_Start ・R_VR_Stop
		11	2. OUTPUT FILES Note update ・R_RTC_Set_BinaryAlarmValue to R_RTC_Set_BinaryAlarmOn
		11, 162, 198	3.2.13. Real-time clock API addition ・r_rtc_callback_periodic
		11	2. OUTPUT FILES API addition ・R_IT_Set_PowerOff
		14, 303, 309	3.2.26. Programmable gain amplifier API addition ・R_PGA_Set_PowerOff
		15	2. OUTPUT FILES API addition ・R_SAUm_Reset
		405, 413, 418	3.2.32. Serial interface IICA Note update ・R_IICAn_StopCondition ・r_iican_callback_master_error ・r_iican_callback_slave_error
1.05	2018.02.01	67-81	3.2.6 Timer RJ API name update ・R_TMR_RJn_Create ・R_TMR_RJn_Start ・R_TMR_RJn_Stop ・R_TMR_RJn_Set_PowerOff ・R_TMR_RJn_Get_PulseWidth ・R_TMR_RJn_Create_UserInit ・r_tmr_rjn_interrupt ・R_TMRJn_Create ・R_TMRJn_Start ・R_TMRJn_Stop ・R_TMRJn_Set_PowerOff ・R_TMRJn_Get_PulseWidth ・R_TMRJn_Create_UserInit ・r_tmrjn_interrupt
		82, 101	3.2.7 Timer RD API addition ・R_TMRD_Set_PowerOff
		215, 222	3.2.16 8-bit interval timer API addition ・R_IT8bitm_set_PowerOff

Rev.	Date	Description	
		Page	Summary
		241, 252, 253	3.2.20 24-bit DS A/D converter with programmable gain instrumentation amplifier API addition ・r_pga_dsad_conversion_interrupt ・r_pga_dsad_scan_interrupt
		296, 307	3.2.25 D/A converter API addition ・R_DACn_Create_UserInit
		455, 461, 462	3.2.36 Data transfer controller API addition ・R_DTCDn_Start ・R_DTCDn_Stop
		524, 529-532	3.2.43 SPI interface API addition ・R_SPI_Create ・R_SPI_Start ・R_SPI_Stop ・R_SPI_Create_UserInit
		596-604	3.2.51 Sampling output timer detector Chapter addition API addition ・R_SMOTD_Create ・R_SMOTD_Start ・R_SMOTD_Stop ・R_SMOTD_Set_PowerOff ・R_SMOTD_Create_UserInit ・r_smotd_counterA_interrupt ・r_smotd_counterB_interrupt ・r_smotd_smpn_interrupt
		605-611	3.2.52 External signal sampler Chapter addition API addition ・R_EXSD_Create ・R_EXSD_Start ・R_EXSD_Stop ・R_EXSD_Set_PowerOff ・R_EXSD_Create_UserInit ・r_exsd_interrupt
		612-626	3.2.53 Serial interface UARTMG Chapter addition API addition ・R_UARTMGn_Create ・R_UARTMGn_Start ・R_UARTMGn_Stop ・R_UARTMGn_Set_PowerOff ・R_UARTMGn_Send ・R_UARTMGn_Receive ・R_UARTMGn_Create_UserInit ・r_uartmgn_interrupt_send ・r_uartmgn_interrupt_receive ・r_uartmgn_interrupt_error ・r_uartmgn_callback_sendend ・r_uartmgn_callback_receiveend ・r_uartmgn_callback_error ・r_uartmgn_callback_softwareoverrun

Rev.	Date	Description	
		Page	Summary
		627-635	3.2.54 Amplifier unit Chapter addition API addition ・R_AMP_Create ・R_PGA1_Start ・R_PGA1_Stop ・R_AMPn_Start ・R_AMPn_Stop ・R_AMP_Set_PowerOn ・R_AMP_Set_PowerOff ・R_AMP_Create_UserInit
		636-642	3.2.55 Data flash libraries Chapter addition API addition ・R_FDL_Create ・R_FDL_Open ・R_FDL_Close ・R_FDL_Write ・R_FDL_Read ・R_FDL_Erase

e^2 studio Code Generator Tool User's Manual:
RL78 API Reference

Publication Date: Rev.1.00 Aug 01, 2014
Rev.1.05 Feb 01, 2018
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.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

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: +44-1628-651-700, Fax: +44-1628-651-804

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, 200333 P. R. China
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, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 5.0

e² studio Code Generator Tool

Renesas Electronics Corporation

R20UT3127EJ0105