User's Manual

Applilet3

Device Driver Configurator

User's Manual: RL78 API Reference

Target Device
RL78 Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2013

Notice

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

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

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

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

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

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

(2012.4)

How to Use This Manual

Readers

This manual understands the function of device driver configurator Applilet3 for RL78 family, and targets the user who designs the application system that uses it.

Purpose

This manual understands the software function of Applilet3 by the user, and has aimed useful as the hardware of the system that uses this and material for the reference to the software development.

Organization

This manual is divided into the following sections.

CHAPTER 1 GENERAL
CHAPTER 2 OUTPUT FILES
CHAPTER 3 API FUNCTIONS

How to Read This Manual

It is assumed that the readers of this manual have general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.
This manual describes the basic setup procedures and how to set switches.

Conventions

Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Decimal ... XXXX
	Hexadecimal ... XXXXH or 0xXXXX

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

TABLE OF CONTENTS

CHAPTER 1 GENERAL ... 5

1.1 Overview ... 5
1.2 Features ... 5

CHAPTER 2 OUTPUT FILES ... 6

2.1 Description ... 6

CHAPTER 3 API FUNCTIONS ... 15

3.1 Overview ... 15
3.2 Function Reference ... 15
3.2.1 Common ... 17
3.2.2 Clock generator ... 23
3.2.3 Port functions ... 38
3.2.4 Timer array unit ... 41
3.2.5 Timer RJ ... 55
3.2.6 Timer RD ... 63
3.2.7 Timer RG ... 73
3.2.8 16-bit timer KB ... 81
3.2.9 16-bit timer KC0 ... 91
3.2.10 Real-time clock ... 98
3.2.11 12-bit interval timer ... 119
3.2.12 16-bit wakeup timer ... 126
3.2.13 Clock output/buzzer output controller ... 133
3.2.14 Watchdog timer ... 139
3.2.15 A/D converter ... 144
3.2.16 D/A converter ... 159
3.2.17 Programmable gain amplifier ... 166
3.2.18 Comparator ... 171
3.2.19 Serial array unit ... 178
3.2.20 Serial array unit 4 (DALI/UART4) ... 217
3.2.21 Asynchronous serial interface LIN-UART (UARTF) ... 230
3.2.22 Serial interface IICA ... 249
3.2.23 LCD controller/driver ... 269
3.2.24 Sound generator ... 278
3.2.25 DMA controller ... 284
3.2.26 DTC ... 293
3.2.27 Event link controller (ELC) ... 299
3.2.28 Interrupt functions ... 303
3.2.29 Key interrupt function ... 312
3.2.30 Voltage detector ... 318

CHAPTER 1 GENERAL

This chapter gives an overview of the Applilet3.

1.1 Overview

The design tool enables you to output the source code (device driver programs, C source files and header files) necessary to control the peripheral hardware functions provided by the microcontroller (clock generator, port functions, etc.) by configuring various information using the GUI.

1.2 Features

The Applilet3 has the following features.

- Code generating function
  The Applilet3 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.

- Project/workspace file generating function
  The Applilet3 can output project and workspace files that can be used in application system integrated development environments (IAR Embedded Workbench).

- Reporting function
  You can output configured information using the Applilet3 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 Applilet3 and the API functions contained in the source code.

CHAPTER 2 OUTPUT FILES

This appendix describes the files output by the Applilet3.

2.1 Description

Below is a list of output file files by the Applilet3.

Table 2-1. Output File List

Peripheral Function	File Name	API Function Name
Common	r_main.c	main R_MAIN_UserInit
	r_systeminit.c	hdwinit __low_level_init R_Systeminit
	r_cg_macrodriver.h	-
	r_cg_userdefine.h	-
	r_option_ca.txt	-
	md_lnk*xxx*_*xx*_xcl	-
Clock generator	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_Set_CRCOn R_CGC_RAMECC_Start R_CGC_RAMECC_Stop R_CGC_StackPointer_Start R_CGC_StackPointer_Stop R_CGC_ClockMonitor_Start R_CGC_ClockMonitor_Stop
	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
	r_cg_cgc.h	-
Port functions	r_cg_port.c	R_PORT_Create
	r_cg_port_user.c	R_PORT_Create_UserInit
	r_cg_port.h	-

Peripheral Function	File Name	API Function Name
Timer array unit	r_cg_timer.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_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_Set_SoftwareTriggerOn
	r_cg_timer_user.c	R_TAUm_Create_UserInit r_taum_channeln_interrupt r_taum_channeln_higher8bits_interrupt
	r_cg_timer.h	-
Timer RJ	r_cg_timer.c	R_TMR_RJ0_Create R_TMR_RJ0_Start R_TMR_RJ0_Stop R_TMR_RJ0_Set_PowerOff R_TMR_RJ0_Get_PulseWidth
	r_cg_timer_user.c	R_TMR_RJ0_Create_UserInit r_tmr_rj0_interrupt
	r_cg_timer.h	-
Timer RD	r_cg_timer.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_cg_timer_user.c	R_TMR_RDn_Create_UserInit r_tmr_rdn_interrupt
	r_cg_timer.h	-
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
	r_cg_timer_user.c	R_TMR_RG0_Create_UserInit r_tmr_rg0_interrupt
	r_cg_timer.h	-

Peripheral Function	File Name	API Function Name
16-bit timer KB	r_cg_timer.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_cg_timer_user.c	R_TMR_KBm_Create_UserInit r_tmr_kbm_interrupt
	r_cg_timer.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
	r_cg_timer_user.c	R_TMR_KC0_Create_UserInit r_tmr_kc0_interrupt
	r_cg_timer.h	-
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_Get_CounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Get_AlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_interrupt r_rtc_callback_constperiod r_rtc_callback_alarm
	r_cg_rtc.h	-
12-bit interval timer	r_cg_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Set_PowerOff
	r_cg_it_user.c	R_IT_Create_UserInit r_it_interrupt
	r_cg_it.h	-

Peripheral Function	File Name	API Function Name
16-bit wakeup timer	r_cg_timer.c	R_WUTM_Create R_WUTM_Start R_WUTM_Stop R_WUTM_Set_PowerOff
	r_cg_timer_user.c	R_WUTM_Create_UserInit r_wutm_interrupt
	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
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit
	r_cg_pclbuz.h	-
Watchdog timer	r_cg_wdt.c	R_WDT_Create R_WDT_Restart
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_interrupt
	r_cg_wdt.h	-
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_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
	r_cg_adc_user.c	R_ADC_Create_UserInit r_adc_interrupt
	r_cg_adc.h	-
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_cg_dac_user.c	R_DAC_Create_UserInit
	r_cg_dac.h	-

Peripheral Function	File Name	API Function Name
Programmable gain amplifier	r_cg_pga.c	R_PGA_Create R_PGA_Start R_PGA_Stop
	r_cg_pga_user.c	R_PGA_Create_UserInit
	r_cg_pga.h	-
Comparator	r_cg_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop R_COMP_Set_PowerOff
	r_cg_comp_user.c	R_COMP_Create_UserInit r_compn_interrupt
	r_cg_comp.h	-

Peripheral Function	File Name	API Function Name
Serial array unit	r_cg_serial.c	R_SAUm_Create R_SAUm_Set_PowerOff R_SAUm_Set_SnoozeOn R_SAUm_Set_SnoozeOff R_UARTn_Create R_UARTn_Start R_UARTn_Stop R_UARTn_Send R_UARTn_Receive R_CSImn_Create R_CSImn_Start R_CSImn_Stop R_CSImn_Send R_CSImn_Receive R_CSImn_Send_Receive R_IICmn_Create R_IICmn_StartCondition R_IICmn_StopCondition R_IICmn_Stop R_IICmn_Master_Send R_IICmn_Master_Receive
	r_cg_serial_user.c	R_SAUm_Create_UserInit r_uartn_interrupt_send r_uartn_interrupt_receive r_uartn_interrupt_error r_uartn_callback_sendend r_uartn_callback_receiveend r_uartn_callback_error r_uartn_callback_softwareoverrun r_csimn_interrupt r_csimn_callback_sendend r_csimn_callback_receiveend r_csimn_callback_error r_iicmn_interrupt r_iicmn_callback_master_sendend r_iicmn_callback_master_receiveend r_iicmn_callback_master_error
	r_cg_serial.h	-

Peripheral Function	File Name	API Function Name
Serial array unit 4 (DALI/ UART4)	r_cg_serial.c	R_DALIn_Create R_DALIn_Start R_DALIn_Stop R_DALIn_Send R_DALIn_Receive
	r_cg_serial_user.c	r_dalin_interrupt_send r_dalin_interrupt_receive r_dalin_interrupt_error r_dalin_callback_sendend r_dalin_callback_receiveend r_dalin_callback_error r_dalin_callback_softwareoverrun
	r_cg_serial.h	-
Asynchronous serial interface LIN-UART (UARTF)	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
	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
	r_cg_serial.h	-

Peripheral Function	File Name	API Function Name
Serial interface IICA	r_cg_serial.c	R_IICAn_Create R_IICAn_StopCondition R_IICAn_Stop 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_cg_serial_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
	r_cg_serial.h	-
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_cg_lcd_user.c	R_LCD_Create_UserInit r_lcd_interrupt
	r_cg_lcd.h	-
Sound generator	r_cg_sg.c	R_SG_Create R_SG_Start R_SG_Stop
	r_cg_sg_user.c	R_SG_Create_UserInit r_sg_interrupt
	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
	r_cg_dmac_user.c	R_DMACn_Create_UserInit R_DMAC_Create_UserInit r_dmacn_interrupt
	r_cg_dmac.h	-

Peripheral Function	File Name	API Function Name
DTC	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff
	r_cg_dtc_user.c	R_DTC_Create_UserInit
	r_cg_dtc.h	-
Event link controller (ELC)	r_cg_elc.c	R_ELC_Create R_ELC_Stop
	r_cg_elc_user.c	R_ELC_Create_UserInit
	r_cg_elc.h	-
Interrupt functions	r_cg_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_INTCLRn_Start R_INTCLRn_Stop
	r_cg_intc_user.c	R_INTC_Create_UserInit r_intcn_interrupt r_intclrn_interrupt
	r_cg_intc.h	-
Key interrupt function	r_cg_intc.c	R_KEY_Create R_KEY_Start R_KEY_Stop
	r_cg_intc_user.c	R_KEY_Create_UserInit r_key_interrupt
	r_cg_intc.h	-
Voltage detector	r_cg_lvd.c	R_LVD_Create R_LVD_InterruptMode_Start
	r_cg_lvd_user.c	R_LVD_Create_UserInit r_lvd_interrupt
	r_cg_lvd.h	-

CHAPTER 3 API FUNCTIONS

This appendix describes the API functions output by the Applilet3.

3.1 Overview

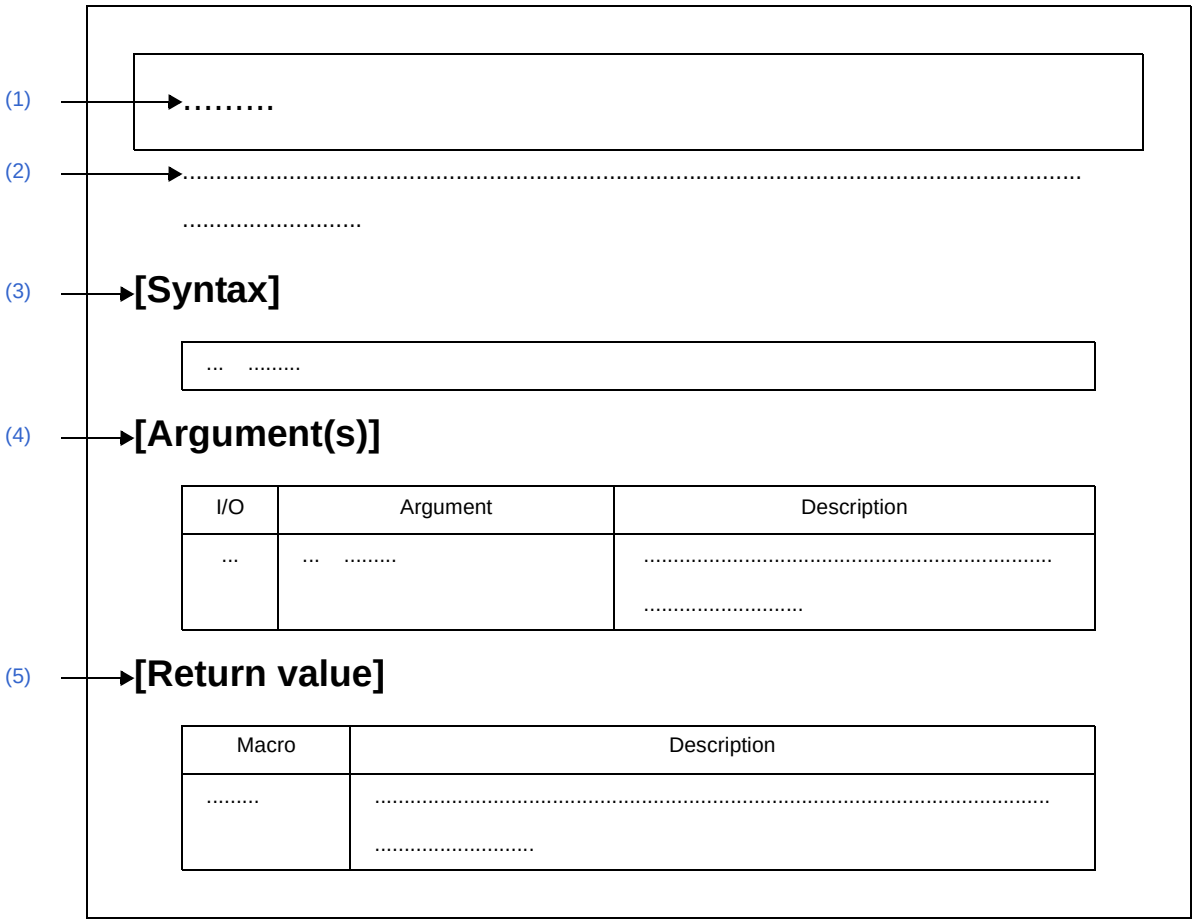
Below are the naming conventions for API functions output by the Applilet3.

- 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 Applilet3, 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 Applilet3 for common use.

Table 3-1. API Functions: [Common]

API Function Name	Function
hdwinit	Performs initialization necessary to control the various hardwares.
__low_level_init	Performs initialization necessary to control the various hardwares.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.

hdwinit

Performs initialization necessary to control the various hardwares.

Remark Call this API function from the startup routine for Renesas Electronics compiler.

[Syntax]

```
void    hdwinit ( void );
```

[Argument(s)]

None.

[Return value]

None.

__low_level_init

Performs initialization necessary to control the various hardwares.

Remark Call this API function from the startup routine for IAR compiler.

[Syntax]

```
void    __low_level_init ( 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, __low_level_init 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 Applilet3 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_Set_CRCOn	Starts the CRC operation function.
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]

```
__interrupt static  void    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]

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

Remark *n* is the interrupt factor number.

[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]

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

Remark *n* is the interrupt factor number.

[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_Set_CRCOn

Starts the CRC operation function.

[Syntax]

```
void    R_CGC_Set_CRCOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

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 Applilet3 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 Timer array unit

Below is a list of API functions output by the Applilet3 for timer array unit use.

Table 3-4. 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_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]

```
__interrupt static  void    r_taum_channeln_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]

```
__interrupt static  void    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.

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*_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.

3.2.5 Timer RJ

Below is a list of API functions output by the Applilet3 for timer RJ use.

Table 3-5. API Functions: [Timer RJ]

API Function Name	Function
R_TMR_RJ0_Create	Performs initialization necessary to control the 16-bit timer RJ0.
R_TMR_RJ0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ0.
r_tmr_rj0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RJ0_Start	Starts the count for 16-bit timer RJ0.
R_TMR_RJ0_Stop	Ends the count for 16-bit timer RJ0.
R_TMR_RJ0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ0.
R_TMR_RJ0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ0.

R_TMR_RJ0_Create

Performs initialization necessary to control the 16-bit timer RJ0.

[Syntax]

```
void    R_TMR_RJ0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ0.

Remark This API function is called as the R_TMR_RJ0_Create callback routine.

[Syntax]

```
void    R_TMR_RJ0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_rj0_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]

```
__interrupt static  void    r_tmr_rj0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Start

Starts the count for 16-bit timer RJ0.

[Syntax]

```
void    R_TMR_RJ0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Stop

Ends the count for 16-bit timer RJ0.

[Syntax]

```
void    R_TMR_RJ0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ0.

Remark Calling this API function changes the 16-bit timer RJ0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RJ0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ0.

Remarks 1. This API function can only be called when the 16-bit timer RJ0 is being used for pulse width measurement mode / pulse period measurement mode.

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_RJ0_Get_PulseWidth ( uint32_t * const active_width );
```

[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 TRJ0IO pin

[Return value]

None.

3.2.6 Timer RD

Below is a list of API functions output by the Applilet3 for timer RD use.

Table 3-6. 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_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]

```
__interrupt static  void    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.

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

3.2.7 Timer RG

Below is a list of API functions output by the Applilet3 for timer RG use.

Table 3-7. 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]

```
__interrupt static  void    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.

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

3.2.8 16-bit timer KB

Below is a list of API functions output by the Applilet3 for 16-bit timer KB use.

Table 3-8. 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_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]

```
__interrupt static  void    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.

3.2.9 16-bit timer KC0

Below is a list of API functions output by the Applilet3 for 16-bit timer KC0 use.

Table 3-9. 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]

```
__interrupt static  void    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.

3.2.10 Real-time clock

Below is a list of API functions output by the Applilet3 for real-time clock use.

Table 3-10. 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 (year, month, weekday, day, hour, minute, second) of the real-time clock.
R_RTC_Get_CounterValue	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
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_AlarmOff	Ends the alarm interrupt function.
R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).
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_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]

```
__interrupt static  void    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.

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 */
    uint8_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_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_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_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)]

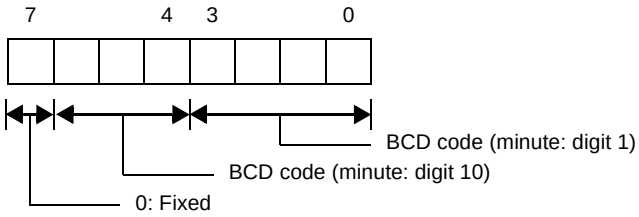
I/O	Argument	Description
I	`rtc_alarm_value_t alarm_val;`	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).

```
typedef struct {
    uint8_t alarmwm;    /* Minute */
    uint8_t alarmwh;    /* Hour */
    uint8_t alarmww;    /* Weekday */
} rtc_alarm_value_t;
```

- alarmwm (Minute)

  Below are shown the meanings of each bit of the structure member alarmwm.



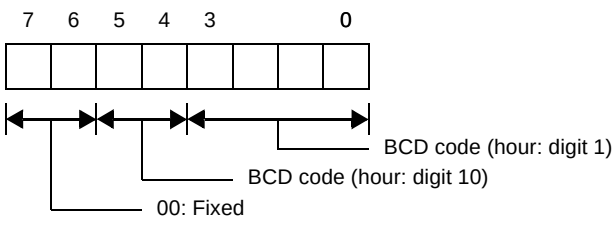
- alarmwh (Hour)

  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.

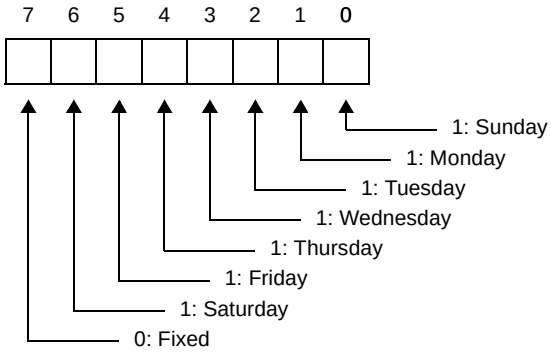
  0: AM

  1: PM



- alarmww (Weekday)

  Below are shown the meanings of each bit of the structure member alarmww.



[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_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.

3.2.11 12-bit interval timer

Below is a list of API functions output by the Applilet3 for 12-bit interval timer use.

Table 3-11. 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_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]

```
__interrupt static  void    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_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.

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.

3.2.12 16-bit wakeup timer

Below is a list of API functions output by the Applilet3 for 16-bit wakeup timer (WUTM) use.

Table 3-12. 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]

```
__interrupt static  void    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.

3.2.13 Clock output/buzzer output controller

Below is a list of API functions output by the Applilet3 for clock output/buzzer output controller use.

Table 3-13. 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.

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.

3.2.14 Watchdog timer

Below is a list of API functions output by the Applilet3 for watchdog timer use.

Table 3-14. 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]

```
__interrupt static  void    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.

3.2.15 A/D converter

Below is a list of API functions output by the Applilet3 for A/D converter use.

Table 3-15. 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_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]

```
__interrupt static  void    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.

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_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).

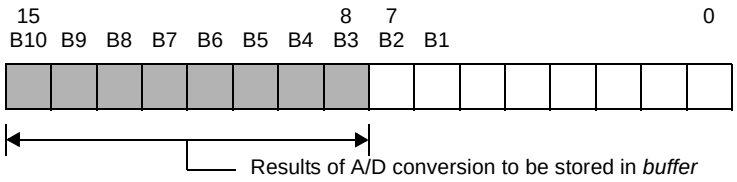
[Syntax]

```
#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.

3.2.16 D/A converter

Below is a list of API functions output by the Applilet3 for D/A converter use.

Table 3-16. 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_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.

3.2.17 Programmable gain amplifier

Below is a list of API functions output by the Applilet3 for programmable gain amplifier use.

Table 3-17. 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_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.

3.2.18 Comparator

Below is a list of API functions output by the Applilet3 for comparator use.

Table 3-18. 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_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]

```
__interrupt static  void    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_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.

3.2.19 Serial array unit

Below is a list of API functions output by the Applilet3 for serial array unit use.

Table 3-19. 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_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*.
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*.

API Function Name	Function
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*_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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    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*.

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*.

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*.

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*.

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*.

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*.

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_IICmn_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_IIC mn _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]

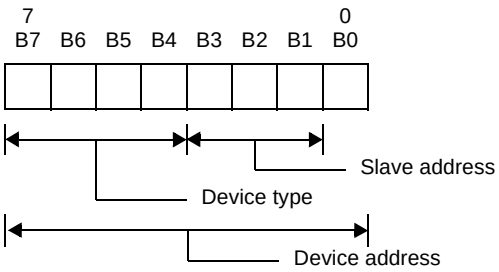
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

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]

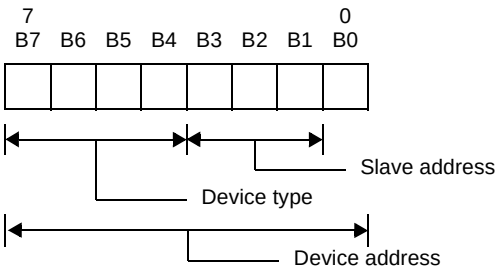
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

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.

3.2.20 Serial array unit 4 (DALI/UART4)

Below is a list of API functions output by the Applilet3 for serial array unit 4 (DALI/UART4) use.

Table 3-20. 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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    r_dalin_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*.

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*.

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_dalin_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_dalin_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_dalin_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_dalin_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.

3.2.21 Asynchronous serial interface LIN-UART (UARTF)

Below is a list of API functions output by the Applilet3 for asynchronous serial interface LIN-UART (UARTF) use.

Table 3-21. 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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    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*.

2. When performing a UARTF transmission, R_UARTFn_Start must be called before this API function is called.

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*.

2. Actual UARTF reception starts after this API function is called, and R_UARTFn_Start is then called.

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.

3.2.22 Serial interface IICA

Below is a list of API functions output by the Applilet3 for serial interface IICA use.

Table 3-22. 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_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_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]

```
__interrupt static  void    r_iican_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_StopCondition

Generates stop conditions.

[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_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_iican_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_iican_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_iican_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

[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_iican_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_iican_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_iican_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

[Return value]

None.

r_iica*n*_callback_getstopcondition

Performs processing in response to detection of stop condition.

[Syntax]

```
static  void    r_iican_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.

3.2.23 LCD controller/driver

Below is a list of API functions output by the Applilet3 for LCD controller/driver use.

Table 3-23. 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_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]

```
__interrupt static  void    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.

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.

3.2.24 Sound generator

Below is a list of API functions output by the Applilet3 for sound generator use.

Table 3-24. 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]

```
__interrupt static  void    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.

3.2.25 DMA controller

Below is a list of API functions output by the Applilet3 for DMA controller use.

Table 3-25. 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]

```
__interrupt static  void    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.

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.

3.2.26 DTC

Below is a list of API functions output by the Applilet3 for DTC use.

Table 3-26. API Functions: [DTC]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the DTC.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the DTC.
R_DTCn_Start	Enables operation of the DTC.
R_DTCn_Stop	Disables operation of the DTC.
R_DTC_Set_PowerOff	Halts the clock supplied to the DTC.

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.

3.2.27 Event link controller (ELC)

Below is a list of API functions output by the Applilet3 for event link controller (ELC) use.

Table 3-27. 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).

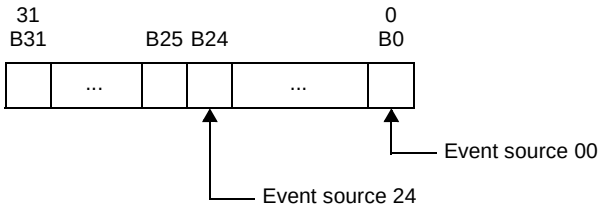
[Syntax]

```
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.

3.2.28 Interrupt functions

Below is a list of API functions output by the Applilet3 for interrupt functions use.

Table 3-28. 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_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]

```
__interrupt static  void    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]

```
__interrupt static  void    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.

3.2.29 Key interrupt function

Below is a list of API functions output by the Applilet3 for key interrupt function use.

Table 3-29. 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]

```
__interrupt static  void    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.

3.2.30 Voltage detector

Below is a list of API functions output by the Applilet3 for voltage detector use.

Table 3-30. 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_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]

```
__interrupt static  void    r_lvd_interrupt ( 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.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 02, 2013	-	First Edition issued

Applilet3 User's Manual:
RL78 API Reference

Publication Date: Rev.1.00 Sep 02, 2013

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2013 Renesas Electronics Corporation and Renesas Solutions Corp.

Colophon 1.3

Applilet3

Renesas Electronics Corporation

R20UT2711EJ0100