User's Manual

CS+ Code Generator Tool

Integrated Development Environment

User's Manual: RH850 API Reference

Target Device

RH850 Family

Target Tool

CS+ V4.00.00

Renesas Electronics
www.renesas.com

Rev.1.01 Mar 2016

Notice

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

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

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

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

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

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

How to Use This Manual

This manual describes the role of the CS+ integrated development environment for developing applications and systems for RH850 family, and provides an outline of its features.

CS+ is an integrated development environment (IDE) for RH850 family, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.

By providing an integrated environment, it is possible to perform all development using just this product, without the need to use many different tools separately.

Readers

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

Purpose

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

Organization

This manual can be broadly divided into the following units.

How to Read This Manual

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

Conventions

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

TABLE OF CONTENTS

1. GENERAL

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

1.1 Overview

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

1.2 Features

Code Generator tool has the following features.

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

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

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

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

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

1.3 About RH850 Code Generator tool

The RH850 Code Generator tool has the following original spec.

- Synchronization processing
  The RH850 Code Generator tool output synchronization processing in Create and Stop of each peripherals. The user has to edit synchronization processing.

- Interrupt vector table
  The RH850 Code Generator tool can not edit interrupt vector table. When using generated interrupt function, please edit interrupt vector table.(Target device : RH850/E1L, E1M-S)

- RH850 Code Generator sample project
  The Code Generator install folder has SampleProjects folder. This folder has sample project.

2. OUTPUT FILE

This appendix describes the files output by the Code Generator.

2.1 Description

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

Table 2.1 Output File List

Peripheral Function	File Name	API Function Name
Common	r_cg_main.c	main R_MAIN_UserInit
	r_cg_systeminit.c	R_SystemInit
	r_cg_macrodriver.h	—
	r_cg_userdefine.h	—
	r_cg_intvector.c	—
Clock controller	r_cg_cgc.c	R_CGC_Create R_CGC_CK_Output_Enable R_CGC_CK_Output_Disable
	r_cg_cgc_user.c	R_CGC_Create_UserInit
	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	—
Interrupt	r_cg_intc.c	R_INTC_Create R_IRQn_Start R_IRQn_Stop R_SINTn_Start R_SINTn_Stop R_SINTn_TriggerOn R_INTPn_Start R_INTPn_Stop
	r_cg_intc_user.c	R_INTC_Create_UserInit r_nmi_interrupt r_irqn_interrupt r_sintn_interrupt r_intpn_interrupt
	r_cg_intc.h	—

Peripheral Function	File Name	API Function Name
DMAC	r_cg_dmac.c	R_DMACn_Create R_DMACn_Suspend R_DMACn_Resume R_DMACnm_Create R_DMACnm_Start R_DMACnm_Stop R_DMACnm_Set_SoftwareTrigger R_DMACnm_Suspend R_DMACnm_Resume
	r_cg_dmac_user.c	R_DMAC_Create_UserInit r_dmacnm_interrupt r_dmacnm_callback_transfer_completion r_dmacnm_callback_transfer_count_match
	r_cg_dmac.h	—
DTS	r_cg_dts.c	R_DTS_Create R_DTS_Suspend R_DTS_Resume R_DTS_All_Stop R_DTSx_y_Stop_Interrupt R_DTSm_Create R_DTSm_Start R_DTSm_Stop R_DTSm_Set_SoftwareTrigger R_DTSm_Suspend R_DTSm_Resume
	r_cg_dts_user.c	R_DTS_Create_UserInit R_DTSm_Create_UserInit r_dtsx_y_transfer_match_interrupt r_dtsx_y_transfer_completion_interrupt
	r_cg_dts.h	—
Clock Serial Interface G	r_cg_csig.c	R_CSIGm_Create R_CSIGm_Start R_CSIGm_Stop R_CSIGm_Send R_CSIGm_Receive
	r_cg_csig_user.c	R_CSIGm_Create_UserInit r_csigm_interrupt_receive r_csigm_interrupt_error r_csigm_interrupt_send r_csigm_callback_receiveend r_csigm_callback_sendend r_csigm_callback_error
	r_cg_csig.h	—

Peripheral Function	File Name	API Function Name
Clock Serial Interface H	r_cg_csih.c	R_CSIHm_Create R_CSIHm_Start R_CSIHm_Stop R_CSIHm_Master_Send R_CSIHm_Master_Receive R_CSIHm_Slave_Send R_CSIHm_Slave_Receive R_CSIHm_Extended_Data_Master_Send R_CSIHm_Extended_Data_Master_Receive R_CSIHm_Extended_Slave_Send R_CSIHm_Extended_Data_Slave_Receive
	r_cg_csih_user.c	R_CSIHm_Create_UserInit r_csihm_interrupt_receive r_csihm_interrupt_error r_csihm_interrupt_send r_csihm_interrupt_jobend r_csihm_callback_receiveend r_csihm_callback_sendend r_csihm_callback_error
	r_cg_csih.h	—
Serial Communication Interface 3	r_cg_sci3.c	R_SCI3m_Create R_SCI3m_Start R_SCI3m_Stop R_SCI3m_Send R_SCI3m_Receive R_SCI3m_Multiprocessor_Send R_SCI3m_Multiprocessor_Receive
	r_cg_sci3_user.c	R_SCI3m_Create_UserInit r_sci3m_interrupt_receive r_sci3m_interrupt_error r_sci3m_interrupt_send r_sci3m_interrupt_sendend r_sci3m_callback_receiveend r_sci3m_callback_sendend r_sci3m_callback_error
	r_cg_sci3.h	—
UART Interface	r_cg_uart.c	R_UARTm_Create R_UARTm_Start R_UARTm_Stop R_UARTm_Send R_UARTm_Receive
	r_cg_uart_user.c	R_UARTm_Create_UserInit r_uartm_interrupt_receive r_uartm_interrupt_error r_uartm_interrupt_send r_uartm_callback_receiveend r_uartm_callback_sendend r_uartm_callback_error
	r_cg_uart.h	—

Peripheral Function	File Name	API Function Name
Window Watchdog Timer	r_cg_wdt.c	R_WDTm_Create R_WDTm_Restart
	r_cg_wdt_user.c	R_WDTm_Create_UserInit r_wdtm_interrupt
	r_cg_wdt.h	—
OS Timer	r_cg_ostm.c	R_OSTMm_Create R_OSTMm_Start R_OSTMm_Stop R_OSTMm_Set_CompareValue
	r_cg_ostm_user.c	R_OSTMm_Create_UserInit r_ostmm_interrupt
	r_cg_ostm.h	—
Advanced Timer Unit IV	r_cg_atuiv.c	R_ATUIV_Common_Create R_ATUIV_Timerkn_Create R_ATUIV_Timerk_OperationOn R_ATUIV_Timerk_OperationOff R_ATUIV_Timerknm_Start R_ATUIV_Timerknm_Stop R_ATUIV_Timerknm_Get_PulseWidth R_ATUIV_Timerknm_Get_CaptureValue R_ATUIV_Timerknm_Set_Compare_Match R_ATUIV_Timerknm_Set_One_Shot_Pulse R_ATUIV_Timerknm_Forced_Compare_Match R_ATUIV_Timerknm_Forced_Output_Compare_Match R_ATUIV_Timerknm_Start_Down_Count R_ATUIV_Timerknm_Get_InputCapturex R_ATUIV_Timerknm_xpin_Output_Normal R_ATUIV_Timerknm_xpin_Output_Low R_ATUIV_Timerknm_xpin_Output_High R_ATUIV_Timerknm_Get_Count R_ATUIV_Timerknm_Get_PWM_Measure_Value R_ATUIV_Timerknm_Get_Measure_Value R_ATUIV_Timerknm_Reset_FIFO
	r_cg_atuiv_user.c	R_ATUIV_Common_Create_UserInit R_ATUIV_Timerkn_Create_UserInit r_atuiv_timerknm_overflow_interrupt r_atuiv_timerknm_interrupt r_atuiv_timerknm_icrnx_interrupt r_atuiv_timerknm_ocrnx_interrupt r_atuiv_timerknm_tcntnx_interrupt r_atuiv_timerknm_cmfnx_interrupt r_atuiv_timerknm_callback_ocrc r_atuiv_timerknm_callback_grc r_atuiv_timerknm_underflow_interrupt r_atuiv_timerknm_comparex_interrupt r_atuiv_timerknm_callback_overflow r_atuiv_timerknm_callback_cycle r_atuiv_timerknm_callback_duty r_atuiv_timerknm_fifo_overflow_interrupt r_atuiv_timerknm_fifo_datafull_interrupt r_atuiv_timerknm_tcntk_overflow_interrupt
	r_cg_atuiv.h	—

Peripheral Function	File Name	API Function Name
Timer Array Unit B	r_cg_taub.c	R_TAUBn_Create R_TAUBn_Channelm_Start R_TAUBn_Channelm_Stop R_TAUBn_Channelm_Get_PulseWidth
	r_cg_taub_user.c	R_TAUBn_Create_UserInit r_taubn_channelm_interrupt
	r_cg_taub.h	—
Timer Array Unit D	r_cg_taud.c	R_TAUDn_Create R_TAUDn_Channelm_Start R_TAUDn_Channelm_Stop R_TAUDn_Channelm_Get_PulseWidth
	r_cg_taud_user.c	R_TAUDn_Create_UserInit r_taudn_channelm_interrupt
	r_cg_taud.h	—
Timer Array Unit J	r_cg_tauj.c	R_TAUJn_Create R_TAUJn_Channelm_Start R_TAUJn_Channelm_Stop R_TAUJn_Channelm_Get_PulseWidth
	r_cg_tauj_user.c	R_TAUJn_Create_UserInit r_taujn_channelm_interrupt
	r_cg_tauj.h	—
Timer Option	r_cg_tapa.c	R_TAPAm_Create R_TAPAm_Start R_TAPAm_Stop R_TAPAm_Trigger_Start R_TAPAm_Trigger_Stop
	r_cg_tapa_user.c	R_TAPAm_Create_UserInit
	r_cg_tapa.h	—
Peripheral Interconnection	r_cg_pic.c	R_PICn_Create R_PICn_Timer_SyncStart
	r_cg_pic_user.c	R_PICn_Create_UserInit
	r_cg_pic.h	—

Peripheral Function	File Name	API Function Name
A/D converter	r_cg_adc.c	R_ADCn_Create R_ADCn_Halt R_ADCn_SetMultiplexerCommand R_ADCn_ScanGroupm_Start R_ADCn_ScanGroupm_GetResult R_ADCn_ScanGroupm_GetFloatingPointDataResult R_ADCn_ScanGroupm_TimerStart R_ADCn_ScanGroupm_TimerStop R_ADCn_ADCSummation_Cannelm_GetResult R_ADCn_ADCSummation_Start R_ADCn_ADCSummation_Stop R_ADC_SyncStart R_ADC_SyncTimerStart R_ADCn_ScanGroupm_OperationOn R_ADCn_TH_Groupx_Start R_ADCn_TH_Sampling_Start
	r_cg_adc_user.c	R_ADCn_Create_UserInit r_adcn_error_interrupt r_adcn_scan_groupm_end_interrupt r_adcn_multiplexer_request_interrupt r_adcn_adc_summation_channelm_end_interrupt
	r_cg_adc.h	–
Delta-Sigma AD converter	r_cg_dsadc.c	R_DSADC_Create R_DSADC_SyncStart R_DSADCm_Start R_DSADCm_Stop R_DSADCm_GetResult
	r_cg_dsadc_user.c	R_DSADC_Create_UserInit r_dsadc_error_interrupt
	r_cg_dsadc.h	–
Digital Filter	r_cg_dfe.c	R_DFE_Create R_DFE_Set_SoftwareData R_DFE_Generate_SoftwareTrigger R_DFE_Channelm_Create R_DFE_Channelm_Enable R_DFE_Channelm_Disable R_DFE_Channelm_GetResult
	r_cg_dfe_user.c	R_DFE_Create_UserInit r_dfe_error_interrupt R_DFE_Channelm_Create_UserInit r_dfe_channelm_interrupt r_dfe_channelm_callback_output_data r_dfe_channelm_callback_condition_match
	r_cg_dfe.h	–
Data CRC	r_cg_dcra.c	R_DCRAn_Create R_DCRAn_Input32bitData R_DCRAn_Input16bitData R_DCRAn_Input8bitData R_DCRAn_GetResult_32bitData R_DCRAn_GetResult_16bitData
	r_cg_dcra_user.c	R_DCRAn_Create_UserInit
	r_cg_dcra.h	–

Peripheral Function	File Name	API Function Name
Real-Time Clock	r_cg_rtca.c	R_RTC_Create R_RTC_Start R_RTC_Stop R_RTC_Set_HourSystem R_RTC_Set_CounterValue R_RTC_Get_CounterValue R_RTC_Set_AlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Get_AlarmValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_1secondInterruptOn R_RTC_Set_1secondInterruptOff R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff
	r_cg_rtca_user.c	R_RTC_Create_UserInit r_rtc_interrupt_periodic r_rtc_interrupt_alarm r_rtc_interrupt_1second
	r_cg_rtca.h	—
Key Return	r_cg_key.c	R_KEY_Create R_KEY_Start R_KEY_Stop
	r_cg_key_user.c	R_KEY_Create_UserInit r_key_interrupt
	r_cg_key.h	—
Stand-By Controller	r_cg_stbc.c	R_STBC_Start_Stop_Mode R_STBC_Prepare_Stop_Mode R_STBC_Start_Deep_Stop_Mode R_STBC_Prepare_Deep_Stop_Mode R_STBC_Deep_Stop_Loop
	r_cg_stbc_user.c	R_STBC_Prepare_Stop_Mode_Set_Peripheral R_STBC_Prepare_Stop_Mode_Set_Interrupt R_STBC_Prepare_Stop_Mode_Set_Clock_Mask R_STBC_Prepare_Stop_Mode_Set_Clock_Source R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral R_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt
	r_cg_stbc.h	—

3. API FUNCTIONS

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

3.1 Overview

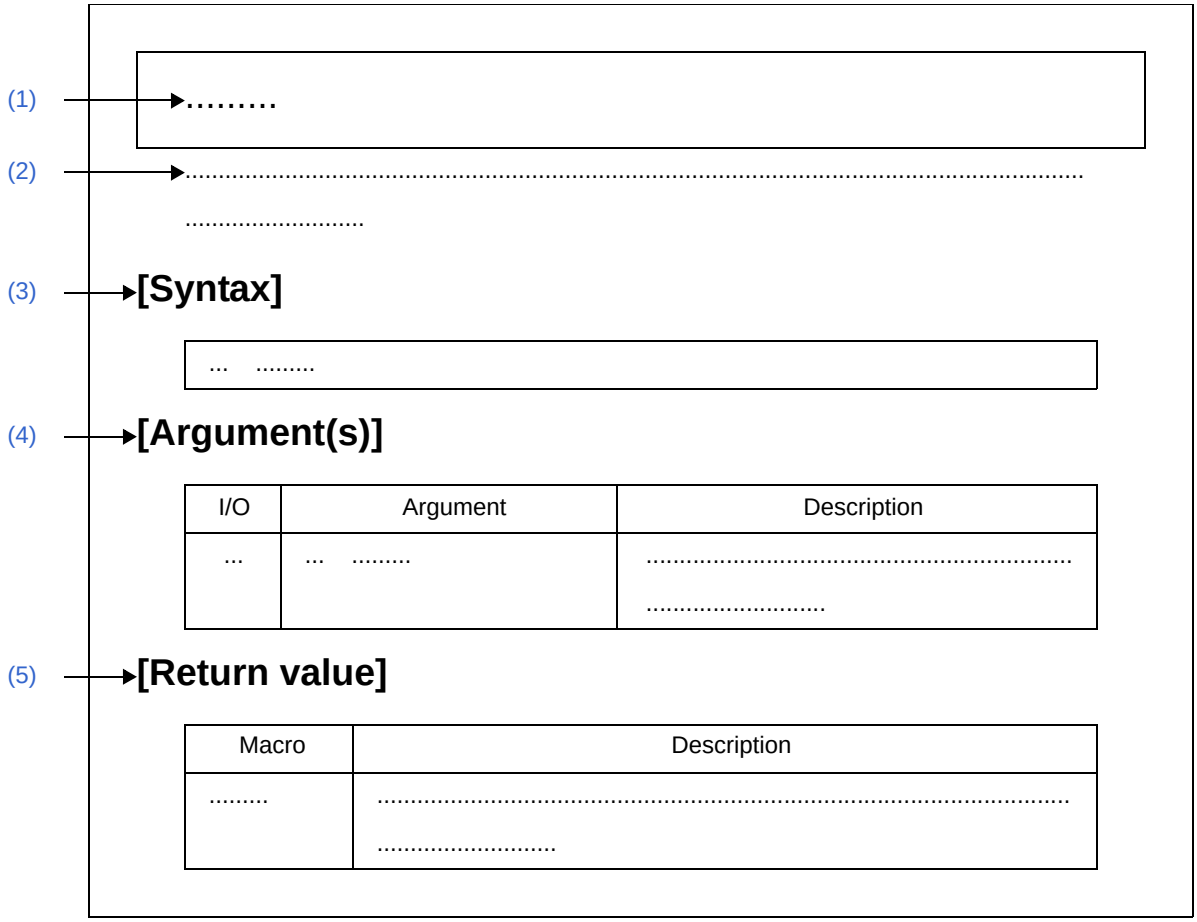
Below are the naming conventions for API functions output by the Code Generator.

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

3.2 Function Reference

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

Figure 3.1 Notation Format of API Functions



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

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

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

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

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

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

(b) Argument
Argument data type

(c) Description
Description of argument

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

Macro	Description
(a)	(b)

(a) Macro
Macro of return value

(b) Description
Description of return value

3.2.1 Common

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

Table 3.1 API Functions: [Common]

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

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.

R_SystemInit

Performs initialization necessary to control the various peripheral functions.

[Syntax]

```
void    R_SystemInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.2 Clock controller

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

Table 3.2 API Functions: [Clock controller]

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-define initialization relating to the clock generator (include reset function, on-chip debug function, etc.).
R_CGC_CK_Output_Enable	Enables output of the CK pin.
R_CGC_CK_Output_Enable	Disables output of the CK pin.

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-define 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_CK_Output_Enable

Enables output of the CK pin.

[Syntax]

```
void R_CGC_CK_Output_Enable(void);
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_CK_Output_Disable

Disables output of the CK pin.

[Syntax]

```
void R_CGC_CK_Output_Disable(void);
```

[Argument(s)]

None.

[Return value]

None.

3.2.3 Port functions

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

Table 3.3 API Functions: [Port functions]

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

R_PORT_Create

Performs initialization necessary to control the port functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the port functions.

Remark This API functions 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 Interrupt

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

Table 3.4 API Functions: [Interrupt]

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_IRQn_Start	Enables the IRQ*n* interrupts.
R_IRQn_Stop	Disables the IRQ*n* interrupts.
R_SINTn_Start	Enables the software interrupts (SINT*n*).
R_SINTn_Stop	Disables the software interrupts (SINT*n*).
R_SINTn_TriggerOn	Software interrupt registers increments the value of a counter.
R_INTPn_Start	Enables the INTP*n* interrupts.
R_INTPn_Stop	Disables the INTP*n* interrupts.
r_nmi_interrupt	Performs processing in response to the NMI interrupt.
r_irqn_interrupt	Performs processing in response to the IRQ*n* interrupt.
r_sintn_interrupt	Performs processing in response to the SINT*n* interrupt.
r_intpn_interrupt	Performs processing in response to the INTP*n* interrupt.

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 functions is called as the R_INTC_Create callback routine.

[Syntax]

```
void    R_INTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IRQ*n*_Start

Enables the IRQ*n* interrupts.

[Syntax]

```
void    R_IRQn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_IRQ*n*_Stop

Disables the IRQ*n* interrupts.

[Syntax]

```
void    R_IRQn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_SINT*n*_Start

Enables the software interrupt (SINT*n*).

[Syntax]

```
void    R_SINTn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_SINT*n*_Stop

Disables the software interrupt (SINT*n*).

[Syntax]

```
void    R_SINTn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_SINT*n*_TriggerOn

Software interrupt registers increments the value of a counter.

[Syntax]

```
void    r_SINTn_TriggerOn ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_nmi_interrupt

Performs processing in response to the NMI interrupt.

[Syntax]

```
void    r_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_irq*n*_interrupt

Performs processing in response to the IRQ*n* interrupt.

[Syntax]

```
void    r_irqn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_sint*n*_interrupt

Performs processing in response to the SINT*n* interrupt.

[Syntax]

```
void    r_sintn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTP*n*_Start

Enables the INTP*n* interrupts.

[Syntax]

```
void    R_INTPn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTP*n*_Stop

Disables the INTP*n* interrupts.

[Syntax]

```
void    R_INTPn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_intp*n*_interrupt

Performs processing in response to the INTP*n* interrupt.

[Syntax]

```
void    r_intpn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

3.2.5 DMAC

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

Table 3.5 API Functions: [DMAC]

API Function Name	Function
R_DMACn_Create	Performs initialization necessary to control the DMAC*n* functions.
R_DMACn_Suspend	Suspend DMAC transfer for all channels.
R_DMACn_Resume	Resume DMAC transfer for all channels.
R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMACn functions.
r_dmacnm_interrupt	Performs processing in response to the DMAC*n* channel*m* interrupt.
r_dmacnm_callback_transfer_completion	Performs processing in response to the DMAC transfer end interrupt.
r_dmacnm_callback_transfer_count_match	Performs processing in response to the DMAC transfer count match interrupt.
R_DMACnm_Create	Performs initialization necessary to control the DMACn channelm functions.
R_DMACnm_Start	Enables the DMAC*n* channel*m* transfer.
R_DMACnm_Stop	Disables the DMAC*n* channel*m* transfer.
R_DMACnm_Set_SoftwareTrigger	Generates the CMAC*n* channel*m* transfer request..
R_DMACnm_Suspend	Suspend DMAC*n* channel*m* transfer.
R_DMACnm_Resume	Resume DMACn channelm transfer.

R_DMAC*n*_Create

Performs initialization necessary to control the DNAC functions.

[Syntax]

```
void    R_DMACn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Suspend

Suspend DMAC transfer for all channels.

[Syntax]

```
void    R_DMACn_Suspend ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Resume

Resume DMAC transfer for all channels.

[Syntax]

```
void    R_DMACn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Create_UserInit

Performs user-defined initialization relating to the DMAC functions.

Remark This API functions is called as the R_DMACn_Create callback routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_dmac*nm*_interrupt

Performs processing in response to the DMAC*n* channel*m* interrupt.

[Syntax]

```
void r_dmacnm_interrupt(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_dmac*nm*_callback_transfer_completion

Performs processing in response to the DMA transfer end interrupt.

[Syntax]

```
void r_dmacnm_callback_transfer_completion(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_dmac*nm*_callback_transfer_count_match

Performs processing in response to the DMA transfer count match interrupt.

[Syntax]

```
void r_dmacnm_callback_transfer_count_match(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Create

Performs initialization necessary to control the DMAC*n* channel*m* functions.

[Syntax]

```
void R_DMACnm_Create(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Start

Enables the DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_DMACnm_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Stop

Disables the DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_DMACnm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Set_SoftwareTrigger

Generates the DMAC*n* channel*m* transfer request.

[Syntax]

```
void    R_DMACnm_Set_SoftwareTrigger ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Suspend

Suspend DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_DMACnm_Suspend ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Resume

Resume DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_DMACnm_Resume ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

3.2.6 DTS

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

Table 3.6 API Functions: [DTS]

API Function Name	Function
R_DTS_Create	Performs initialization necessary to control the DTS functions.
R_DTS_Create_UserInit	Performs user-defined initialization relating to the DTS functions.
R_DTS_Suspend	Suspend DTS transfer.
R_DTS_Resume	Resume DTS transfer/
R_DTS_All_Stop	Stop all DTS transfer.
R_DTSx_y_Stop_Interrupt	Stop DTS transfer interrupt.
R_DTSm_Create	Performs initialization necessary to control the DTS channel *m* functions.
R_DTSm_Start	Enables the DTS channel *m* transfer.
R_DTSm_Stop	Disables the DTS channel *m* transfer.
R_DTSm_Set_SoftwareTrigger	Generates the DTS channel *m* transfer request.
R_DTSm_Suspend	Suspend DTS channel *m* transfer.
R_DTSm_Resume	Resume DTS channel *m* transfer.
R_DTSm_Create_UserInit	Performs user-defined initialization relating to the DTS channel *m* functions.
r_dtsx_y_transfer_match_interrupt	Performs processing in response to the DTS transfer match interrupt.
r_dtsx_y_transfer_completion_interrupt	Performs processing in response to the DTS transfer completion interrupt.

R_DTS_Create

Performs initialization necessary to control the DTS functions.

[Syntax]

```
void    R_DTS_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTS_Create_UserInit

Performs user-defined initialization relating to the DTS functions.

Remark This API functions is called as the R_DTS_Create callback routine.

[Syntax]

```
void    R_DTS_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTS_Suspend

Suspend DTS transfer.

[Syntax]

```
void    R_DTS_Suspend ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTS_Resume

Resume DTS transfer.

[Syntax]

```
void    R_DTS_Resume ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTS_All_Stop

Stop all DTS transfer.

[Syntax]

```
void    R_DTS_All_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTS*x*_*y*_Stop_Interrupt

Stop DTS transfer interrupt (Range is channel *x* to channel *y*).

[Syntax]

```
void    R_DTSx_y__Stop_Interrupt ( void );
```

Remark *x* and *y* is channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Create

Performs initialization necessary to control the DTS channel *m* functions.

[Syntax]

```
void    R_DTSm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Start

Enables the DTS channel*m* transfer.

[Syntax]

```
void    R_DTSm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Stop

Disables the DTS channel *m* transfer.

[Syntax]

```
void    R_DTSm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Set_SoftwareTrigger

Generates the DTS channel *m* transfer request.

[Syntax]

```
void    R_DTSm_Set_SoftwareTrigger ( void );
```

Remark m is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Suspend

Suspend DTS channel *m* transfer.

[Syntax]

```
void    R_DTSm_Suspend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Resume

Resume DTS channel *m* transfer.

[Syntax]

```
void    R_DTSm_Resume ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTS*m*_Create_UserInit

Performs user-defined initialization relating to the DTS channel *m* functions.

Remark This API functions is called as the R_DTSm_Create callback routine.

[Syntax]

```
void    R_DTSm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dts*x*_*y*_transfer_match_interrupt

Performs processing in response to the DTS transfer match interrupt (Range is channel *x* to channel *y*).

[Syntax]

```
void    r_dtsx_y_transfer_match_interrupt ( void );
```

Remark *x* and *y* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dts*x*_*y*_transfer_completion_interrupt

Performs processing in response to the DTS transfer completion interrupt (Range is channel *x* to channel *y*).

[Syntax]

```
void    r_dtsx_y_transfer_completion_interrupt ( void );
```

Remark *x* and *y* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.7 Clock Serial Interface G

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

Table 3.7 API Functions: [Clock serial interface G]

API Function Name	Function
R_CSIGm_Create	Performs initialization necessary to control the clock serial interface G functions.
R_CSIGm_Create_UserInit	Performs user-defined initialization relating to the clock serial interface G functions.
r_csigm_interrupt_receive	Performs processing in response to the CSIG reception interrupt.
r_csigm_interrupt_error	Performs processing in response to the CSIG error interrupt.
r_csigm_interrupt_send	Performs processing in response to the CSIG communication interrupt.
r_csigm_callback_receiveend	Performs processing in response to the CSIG reception interrupt.
r_csigm_callback_sendend	Performs processing in response to the CSIG communication interrupt.
r_csigm_callback_error	Performs processing in response to the CSIG error interrupt.
R_CSIGm_Start	Sets CSIG communication to standby mode.
R_CSIGm_Stop	Ends CSIG communication.
R_CSIGm_Send	Start CSIG data transmission.
R_CSIGm_Receive	Start CSIG data reception.

R_CSIG*m*_Create

Performs initialization necessary to control the clock serial interface G functions.

[Syntax]

```
void    R_CSIGm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIG*m*_Create_UserInit

Performs user-defined initialization relating to the clock serial interface G functions.

Remark This API functions is called as theR_CSIGm_Create callback routine.

[Syntax]

```
void    R_CSIGm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_interrupt_receive

Performs processing in response to the CSIG reception interrupt.

[Syntax]

```
void    r_csigm_interrupt_receive ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_interrupt_error

Performs processing in response to the CSIG error interrupt.

[Syntax]

```
void    r_csigm_interrupt_error ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_interrupt_send

Performs processing in response to the CSIG communication interrupt.

[Syntax]

```
void    r_csigm_interrupt_send ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_callback_receiveend

Performs processing in response to the CSIG reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_csigm_interrupt_receive corresponding to the CSIG reception interrupt.

[Syntax]

```
void    r_csigm_callback_receiveend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_callback_sendend

Performs processing in response to the CSIG communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_csigm_interrupt_send corresponding to the CSIG communication interrupt.

[Syntax]

```
void    r_csigm_callback_sendend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csig*m*_callback_error

Performs processing in response to the CSIG error interrupt.

Remark This API function is called as the callback routine of interrupt process r_csigm_interrupt_errorcorresponding to the CSIG error interrupt.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint8_t err_type;`	Trigger for CSIG error interrupt 0000x0x1B : Overrun error 0000x01xB : Parity error 000010xxB : Data consistency check error

[Return value]

None.

R_CSIG*m*_Start

Sets CSIG communication to standby mode.

[Syntax]

```
void    R_CSIGm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIG*m*_Stop

Ends CSIG communication.

[Syntax]

```
void    R_CSIGm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIG*m*_Send

Start CSIG data transmission.

Remark 1. This API function repeats the 2 byte-level CSIG transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. R_CSIGm_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * 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_CSIG*m*_Receive

Start CSIG data reception.

Remark 1. This API function performs 2 byte-level CSIG reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_CSIGm_Start is then called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * 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

3.2.8 Clock Serial Interface H

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

Table 3.8 API Functions: [Clock serial interface H]

API Function Name	Function
R_CSIHm_Create	Performs initialization necessary to control the clock serial interface H functions.
R_CSIHm_Create_UserInit	Performs user-defined initialization relating to the clock serial interface H functions.
r_csihm_interrupt_receive	Performs processing in response to the CSIH reception interrupt.
r_csihm_interrupt_error	Performs processing in response to the CSIH error interrupt.
r_csihm_interrupt_send	Performs processing in response to the CSIH communication interrupt.
r_csihm_interrupt_jobend	Performs processing in response to the CSIH job end interrupt.
r_csihm_callback_receiveend	Performs processing in response to the CSIH reception interrupt.
r_csihm_callback_sendend	Performs processing in response to the CSIH communication interrupt.
r_csihm_callback_error	Performs processing in response to the CSIH error interrupt.
R_CSIHm_Start	Sets CSIH communication to standby mode.
R_CSIHm_Stop	Ends CSIH communication.
R_CSIHm_Master_Send	Start CSIH data transmission by master mode.
R_CSIHm_Master_Receive	Start CSIH data reception by master mode.
R_CSIHm_Slave_Send	Start CSIH data transmission by slave mode.
R_CSIHm_Slave_Receive	Start CSIH data reception by slave mode.
R_CSIHm_Extended_Data_Master_Send	Start CSIH extended data transmission by master mode.
R_CSIHm_Extended_Data_Master_Receive	Start CSIH extended data reception by master mode.
R_CSIHm_Extended_Slave_Send	Start CSIH extended data transmission by slave mode.
R_CSIHm_Extended_Data_Slave_Receive	Start CSIH extended data reception by slave mode.

R_CSIH*m*_Create

Performs initialization necessary to control the clock serial interface H functions.

[Syntax]

```
void    R_CSIHm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIH*m*_Create_UserInit

Performs user-defined initialization relating to the clock serial interface H functions.

Remark This API functions is called as theR_CSIHm_Create callback routine.

[Syntax]

```
void    R_CSIHm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_interrupt_receive

Performs processing in response to the CSIH reception interrupt.

[Syntax]

```
void    r_csihm_interrupt_receive ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_interrupt_error

Performs processing in response to the CSIH error interrupt.

[Syntax]

```
void    r_csihm_interrupt_error ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_interrupt_send

Performs processing in response to the CSIH communication interrupt.

[Syntax]

```
void    r_csihm_interrupt_send ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_interrupt_jobend

Performs processing in response to the CSIH job end interrupt.

[Syntax]

```
void    r_csihm_interrupt_jobend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_callback_receiveend

Performs processing in response to the CSIH reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_csihm_interrupt_receive corresponding to the CSIH reception interrupt.

[Syntax]

```
void    r_csihm_callback_receiveend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_callback_sendend

Performs processing in response to the CSIH communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_csihm_interrupt_send corresponding to the CSIH communication interrupt.

[Syntax]

```
void    r_csihm_callback_sendend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csih*m*_callback_error

Performs processing in response to the CSIH error interrupt.

Remark This API function is called as the callback routine of interrupt process r_csihm_interrupt_errorcorresponding to the CSIH error interrupt.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    r_csih m_callback_error ( uint8_t err_type );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint8_t err_type;`	Trigger for CSIH error interrupt 0000x0x1B : Overrun error 0000x01xB : Parity error 000010xxB : Data consistency check error

[Return value]

None.

R_CSIH*m*_Start

Sets CSIH communication to standby mode.

[Syntax]

```
void    R_CSIHm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIH*m*_Stop

Ends CSIH communication.

[Syntax]

```
void    R_CSIHm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSIH*m*_Master_Send

Start CSIH data transmission by master mode.

Remark 1. This API function repeats the 2 byte-level CSIH transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. R_CSIHm_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS    R_CSIHm_Master_Send ( const uint16_t * tx_buf, uint16_t tx_num,
uint32_t chipId );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
I	`uint32_t chipId;`	Set chip select

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSIH*m*_Master_Receive

Start CSIH data reception by master mode.

Remark 1. This API function performs 2 byte-level CSIH reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_CSIHm_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS R_CSIHm_Master_Receive ( const uint16_t * rx_buf, uint16_t rx_num,
uint32_t chipId );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * rx_buf;`	Pointer to a buffer to store the received data
I	`uint16_t rx_num;`	Total amount of data to receive
I	`uint32_t chipId;`	Set chip select

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSIH*m*_Slave_Send

Start CSIH data transmission by slave mode.

Remark 1. This API function repeats the 2 byte-level CSIH transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. R_CSIHm_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * 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_CSIH*m*_Slave_Receive

Start CSIH data reception by slave mode.

Remark 1. This API function performs 2 byte-level CSIH reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_CSIHm_Start is then called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * rx_buf;`	Pointer to a buffer storing 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_CSIH*m*_Extended_Data_Master_Send

Start CSIH extended data transmission by master mode.

Remark 1. This API function repeats the 2 byte-level CSIH transmission from the buffer specified in argument *tx_buf* the number of bits specified in argument *tx_bit_num*.

Remark 2. R_CSIHm_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS    R_CSIHm_Extended_Data_Master_Send ( const uint16_t * tx_buf, uint16_t
tx_bit_num, uint32_t chipId );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_bit_num;`	Number of bits to send
I	`uint32_t chipId;`	Set chip select

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSIH*m*_Extended_Data_Master_Receive

Start CSIH extended data reception by master mode.

Remark 1. This API function performs 2 byte-level CSIH reception the number of bots specified by the argument *rx_bit_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_CSIHm_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS R_CSIHm_Master_Receive ( const uint16_t * rx_buf, uint16_t rx_bit_num,
uint32_t chipId );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * rx_buf;`	Pointer to a buffer to store the received data
I	`uint16_t rx_bit_num;`	Number of bits to receive
I	`uint32_t chipId;`	Set chip select

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSIH*m*_Extended_Slave_Send

Start CSIH extended data transmission by slave mode.

Remark 1. This API function repeats the 2 byte-level CSIH transmission from the buffer specified in argument *tx_buf* the number of bits specified in argument *tx_num*.

Remark 2. R_CSIHm_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS    r_CSIHm_Slave_Send ( const uint16_t * tx_buf, uint16_t tx_bit_num);
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_bit_num;`	Number of bits to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSIH*m*_Extended_Data_Slave_Receive

Start CSIH extended data reception by slave mode.

Remark 1. This API function performs 2 byte-level CSIH reception the number of bits specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_CSIHm_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS    R_CSIHm_Slave_Receive ( const uint16_t * rx_buf, uint16_t
rx_bit_num );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * rx_buf;`	Pointer to a buffer storing the received data
I	`uint16_t rx_bit_num;`	Number of bits to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.2.9 Serial Communication Interface 3

Below is a list of API functions output by the Code Generator for serial communication interface 3 use.

Table 3.9 API Functions: [Serial communication interface 3]

API Function Name	Function
R_SCI3m_Create	Performs initialization necessary to control the serial communication interface 3 functions.
R_SCI3m_Create_UserInit	Performs user-defined initialization relating to the serial communication interface 3 functions.
r_sci3m_interrupt_receive	Performs processing in response to the SCI reception interrupt.
r_sci3m_interrupt_error	Performs processing in response to the SCI error interrupt.
r_sci3m_interrupt_send	Performs processing in response to the SCI transmit interrupt.
r_sci3m_interrupt_sendend	Performs processing in response to the SCI transmit end interrupt.
r_sci3m_callback_receiveend	Performs processing in response to the SCI reception interrupt.
r_sci3m_callback_sendend	Performs processing in response to the SCI transmit end interrupt.
r_sci3m_callback_error	Performs processing in response to the SCI error interrupt.
R_SCI3m_Start	Sets SCI communication to standby mode.
R_SCI3m_Stop	Ends SCI3 communication.
R_SCI3m_Send	Starts SCI3 data transmission.
R_SCI3m_Receive	Starts SCI3 data reception.
R_SCI3m_Multiprocessor_Send	Starts SCI3 data transmission by multiprocessor mode.
R_SCI3m_Multiprocessor_Receive	Starts SCI3 data reception by multiprocessor mode.

R_SCI3*m*_Create

Performs initialization necessary to control the serial communication interface 3 functions.

[Syntax]

```
void    R_SCI3m_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI3*m*_Create_UserInit

Performs user-defined initialization relating to the serial communication interface 3 functions.

Remark This API functions is called as the R_SCI3m_Create callback routine.

[Syntax]

```
void    R_SCI3m_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_interrupt_receive

Performs processing in response to the SCI reception interrupt.

[Syntax]

```
void    r_sci3m_interrupt_receive ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_interrupt_error

Performs processing in response to the SCI error interrupt.

[Syntax]

```
void    r_sci3m_interrupt_error ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_interrupt_send

Performs processing in response to the SCI transmit interrupt.

[Syntax]

```
void    r_sci3m_interrupt_send ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_interrupt_sendend

Performs processing in response to the SCI transmit end interrupt.

[Syntax]

```
void    r_sci3m_interrupt_sendend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_callback_receiveend

Performs processing in response to the SCI reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_sci3m_interrupt_receive corresponding to the CSI reception interrupt.

[Syntax]

```
void    r_sci3m_callback_receiveend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_callback_sendend

Performs processing in response to the SCI transmit interrupt.

Remark This API function is called as the callback routine of interrupt process r_sci3m_interrupt_sendend corresponding to the SCI transmit interrupt.

[Syntax]

```
void    r_sci3m_callback_sendend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci3*m*_callback_error

Performs processing in response to the SCI error interrupt.

Remark This API function is called as the callback routine of interrupt process r_sci3m_interrupt_error corresponding to the SCI error interrupt.

[Syntax]

```
void    r_SCI3m_callback_error ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI3*m*_Start

Sets SCI3 communication to standby mode.

[Syntax]

```
void    R_SCI3m_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI3*m*_Stop

Ends SCI3 communication.

[Syntax]

```
void    R_SCI3m_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI3*m*_Send

Starts SCI3 data transmission.

Remark 1. This API function repeats the byte-level SCI3 transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. R_SCI3m_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCI3m_Send ( const uint8_t * tx_buf, uint16_t tx_num );
```

Remark *n* は , is the channel number を意味します。

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint8_t * 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_SCI3*m*_Receive

Starts SCI3 data reception.

Remark 1. This API function performs byte-level SCI3 reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_SCI3m_Start is then called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCI3m_Receive ( uint8_t * rx_buf, uint16_t rx_num );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint8_t * rx_buf;`	Pointer to a buffer storing 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_SCI3*m*_Multiprocessor_Send

Starts SCI3 data transmission by multiprocessor mode.

Remark 1. This API function repeats the byte-level SCI3 transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. R_SCI3m_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS R_SCI3m_Multiprocessor_Send( const uint8_t * tx_buf, uint16_t tx_num,
uint8_t rx_id);
```

Remark *m* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_SCI3*m*_Multiprocessor_Receive

Starts SCI3 data reception by multiprocessor mode.

Remark 1. This API function performs byte-level SCI3 reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. Starts after this API function is called, and R_SCI3m_Start is then challed.

[Syntax]

```
MD_STATUS R_SCI3m_Multiprocessor_Receive( uint8_t * rx_buf, uint16_t rx_num, uint8_t rx_id);
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint8_t * const rx_buf;`	Pointer to a buffer storing the received data
I	`uint16_t rx_num;`	Total amount of data to receive
I	`uint8_t rx_id;`	Reception id

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.2.10 UART Interface

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

Table 3.10 API Functions: [UART interface]

API Function Name	Function
R_UARTm_Create	Performs initialization necessary to control the UART interface functions.
R_UARTm_Create_UserInit	Performs user-defined initialization relating to the UART interface functions.
r_uartm_interrupt_receive	Performs processing in response to the UART reception interrupt.
r_uartm_interrupt_error	Performs processing in response to the UART error interrupt.
r_uartm_interrupt_send	Performs processing in response to the UART communication interrupt.
r_uartm_callback_receiveend	Performs processing in response to the UART reception interrupt.
r_uartm_callback_sendend	Performs processing in response to the UART communication interrupt.
r_uartm_callback_error	Performs processing in response to the UART error interrupt.
R_UARTm_Start	Sets UART communication to standby mode.
R_UARTm_Stop	Ends UART communication.
R_UARTm_Send	Start UART data transmission.
R_UARTm_Receive	Start UART data reception.

R_UART*m*_Create

Performs initialization necessary to control the UART interface functions.

[Syntax]

```
void    R_UARTm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*m*_Create_UserInit

Performs user-defined initialization relating to the UART interface functions.

Remark This API functions is called as theR_UARTm_Create callback routine.

[Syntax]

```
void    R_UARTm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_interrupt_receive

Performs processing in response to the UART reception interrupt.

[Syntax]

```
void    r_uartm_interrupt_receive ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_interrupt_error

Performs processing in response to the UART error interrupt.

[Syntax]

```
void    r_uartm_interrupt_error ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_interrupt_send

Performs processing in response to the UART communication interrupt.

[Syntax]

```
void    r_uartm_interrupt_send ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_callback_receiveend

Performs processing in response to the UART reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_uartm_interrupt_receive corresponding to the UART reception interrupt.

[Syntax]

```
void    r_uartm_callback_receiveend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_callback_sendend

Performs processing in response to the UART communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_uartm_interrupt_send corresponding to the UART communication interrupt.

[Syntax]

```
void    r_uartm_callback_sendend ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*m*_callback_error

Performs processing in response to the UART error interrupt.

Remark This API function is called as the callback routine of interrupt process r_uartm_interrupt_errorcorresponding to the UART error interrupt.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint8_t err_type;`	Trigger for UART error interrupt 0x00xx01B : Bit error 0x00x10xB : Overrun error 0x001x0xB : Framing error 0100xx0xB : Parity error

[Return value]

None.

R_UART*m*_Start

Sets UART communication to standby mode.

[Syntax]

```
void    R_UARTm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*m*_Stop

Ends UART communication.

[Syntax]

```
void    R_UARTm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*m*_Send

Start UART data transmission.

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

Remark 2. R_UARTm_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * 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*m*_Receive

Start UART data reception.

Remark 1. This API function performs 1 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*.

Remark 2. Starts after this API function is called, and R_UARTm_Start is then called.

[Syntax]

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

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`const uint16_t * 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

3.2.11 Window Watchdog Timer

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

Table 3.11 API Functions: [Window watchdog timer]

API Function Name	Function
R_WDTm_Create	Performs initialization necessary to control the watchdog timer functions.
R_WDTm_Create_UserInit	Performs user-defined initialization relating to the watchdog timer functions.
r_wdtm_interrupt	Performs processing in response to the interval interrupt.
R_WDTm_Restart	Clears the watchdog timer counter and resumes counting.

R_WDT*m*_Create

Performs initialization necessary to control the watchdog timer functions.

[Syntax]

```
void    R_WDTm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_WDT*m*_Create_UserInit

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

Remark This API functions is called as theR_WDTm_Create callback routine.

[Syntax]

```
void    R_WDTm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_wdt*m*_interrupt

Performs processing in response to the interval interrupt.

[Syntax]

```
void    r_wdtm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_WDT*m*_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```
void    R_WDTm_Restart ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.12 OS Timer

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

Table 3.12 API Functions: [OS timer]

API Function Name	Function
R_OSTMm_Create	Performs initialization necessary to control the OS timer functions.
R_OSTMm_Create_UserInit	Performs user-defined initialization relating to the OS timer functions.
r_ostmm_interrupt	Performs processing in response to the OS timer interrupt.
R_OSTMm_Start	Start OS timer count.
R_OSTMm_Stop	Stop OS timer count.
R_OSTMm_Set_CompareValue	In interval timer mode, set start value of the down-counter. In free-running comparison mode, set value for comparison.

R_OSTM*m*_Create

Performs initialization necessary to control the OS timer functions.

[Syntax]

```
void    R_OSTMm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OSTM*m*_Create_UserInit

Performs user-defined initialization relating to the OS timer functions.

Remark This API functions is called as theR_OSTMm_Create callback routine.

[Syntax]

```
void    R_OSTMm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_ostm*m*_interrupt

Performs processing in response to the OS timer interrupt.

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

[Syntax]

```
void    r_ostmm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OSTM*m*_Start

Start OS timer count.

[Syntax]

```
void    R_OSTMm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OSTM*m*_Stop

Stop OS timer count.

[Syntax]

```
void    R_OSTMm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OSTM*m*_Set_CompareValue

In interval timer mode, set start value of the down-counter.
In free-running comparison mode, set value for comparison.

[Syntax]

```
void    R_OSTMm_Set_CompareValue ( uint32 value );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	`uint32 value;`	Start value of the down-counter or comparison value

[Return value]

None.

3.2.13 Advanced Timer Unit IV

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

Table 3.13 API Functions: [Advanced timer unit IV]

API Function Name	Function
R_ATUIV_Common_Create	Performs initialization necessary to control the advanced timer unit IV functions.
R_ATUIV_Common_Create_UserInit	Performs user-defined initialization relating to the advanced timer unit IV functions.
R_ATUIV_Timerkn_Create	Performs initialization necessary to control the each timer functions.
R_ATUIV_Timerkn_Create_UserInit	Performs user-defined initialization relating to the each timer functions.
r_atuiv_timerknm_overflow_interrupt	Performs processing in response to the overflow interrupt.
r_atuiv_timerknm_interrupt	Performs processing in response to the timer interrupt (Input capture, compare match).
r_atuiv_timerknm_icrnx_interrupt	Performs processing in response to the ICRnx register interrupt.
r_atuiv_timerknm_ocrnx_interrupt	Performs processing in response to the OCRnx register interrupt.
r_atuiv_timerknm_tcntnx_interrupt	Performs processing in response to the TCNTnx interrupt.
r_atuiv_timerknm_cmfnx_interrupt	Performs processing in response to the CMFnx register interrupt.
r_atuiv_timerknm_callback_ocrc	Performs processing in response to the OCRCnm register interrupt.
r_atuiv_timerknm_callback_grc	Performs processing in response to the GRCnm register interrupt.
r_atuiv_timerknm_underflow_interrupt	Performs processing in response to the underflow interrupt.
r_atuiv_timerknm_comparex_interrupt	Performs processing in response to the compare match compare register and counter interrupt
r_atuiv_timerknm_callback_overflow	Performs processing in response to the overflow interrupt.
r_atuiv_timerknm_callback_cycle	Performs processing in response to the cycle match interrupt.
r_atuiv_timerknm_callback_duty	Performs processing in response to the duty match interrupt.
r_atuiv_timerknm_fifo_overflow_interrupt	Performs processing in response to the FIFO overflow interrupt.
r_atuiv_timerknm_fifo_datafull_interrupt	Performs processing in response to the FIFO data full interrupt.
r_atuiv_timerknm_tcntk_overflow_interrupt	Performs processing in response to the counter overflow interrupt.
R_ATUIV_Timerk_OperationOn	Enables operation of each timer.
R_ATUIV_Timerk_OperationOff	Disables operation of each timer.
R_ATUIV_Timerknm_Start	Starts each timer count.
R_ATUIV_Timerknm_Stop	Ends each timer count.
R_ATUIV_Timerknm_Get_PulseWidth	Reads the input pulse width of the timer.
R_ATUIV_Timerknm_Get_CaptureValue	Reads the input capture register of the timer.
R_ATUIV_Timerknm_Set_Compare_Match	Sets the compare match register.
R_ATUIV_Timerknm_Set_One_Shot_Pulse	Renewal the one shot pulse register.
R_ATUIV_Timerknm_Forced_Compare_Match	Performs forced compare match.

API Function Name	Function
R_ATUIV_Timerknm_Forced_Output_Compare_Match	Performs forced output compare match.
R_ATUIV_Timerknm_Start_Down_Count	Starts the down count.
R_ATUIV_Timerknm_Get_InputCapturex	Reads the input capture register of the timer.
R_ATUIV_Timerknm_xpin_Output_Normal	Output of pin provides normal output.
R_ATUIV_Timerknm_xpin_Output_Low	Output of pin provides low output.
R_ATUIV_Timerknm_xpin_Output_High	Output of pin provides high output.
R_ATUIV_Timerknm_Get_Count	Reads the counter value of measurement.
R_ATUIV_Timerknm_Get_PWM_Measure_Value	Reads the PWM wave of measurement.
R_ATUIV_Timerknm_Get_Measure_Value	Reads the value of measurement (number of edges, off-state duty cycle, PWM cycle, edge input time).
R_ATUIV_Timerknm_Reset_FIFO	Resets FIFO to the idle state.

R_ATUIV_Common_Create

Performs initialization necessary to control the advanced timer unit IV functions.

[Syntax]

```
void    R_ATUIV_Common_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Common_Create_UserInit

Performs user-defined initialization relating to the advanced timer unit IV functions.

Remark This API functions is called as the R_ATUIV_Common_Create callback routine.

[Syntax]

```
void    R_ATUIV_Common_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*kn*_Create

Performs initialization necessary to control the each timer functions.

[Syntax]

```
void    R_ATUIV_Timerk_Create ( void );
```

```
void    R_ATUIV_Timerkn_Create ( void );
```

Remark *k* is the timer kind, *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*kn*_Create_UserInit

Performs user-defined initialization relating to the each timer functions.

Remark This API functions is called as the R_ATUIV_Timerkn_Create callback routine.

[Syntax]

```
void    R_ATUIV_Timerk_Create_UserInit ( void );
```

```
void    R_ATUIV_Timerkn_Create_UserInit ( void );
```

Remark *k* is the timer kind, *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_overflow_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
void    r_atuiv_timerknm_overflow_interrupt ( void );
```

```
void    r_atuiv_timerknm_overflow1_interrupt ( void );
void    r_atuiv_timerknm_overflow2_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_interrupt

Performs processing in response to the timer interrupt (Input capture, compare match).

[Syntax]

```
void    r_atuiv_timerknm_interrupt ( void );
```

```
void    r_atuiv_timerkn_channelm_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_icr*nx*_interrupt

Performs processing in response to the ICR*nx* register interrupt.

[Syntax]

```
void    r_atuiv_timerknm_icrnx_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the register number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_ocr*nx*_interrupt

Performs processing in response to the OCR*nx* register interrupt.

[Syntax]

```
void    r_atuiv_timerknm_ocrnx_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the register number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_tcnt*nx*_interrupt

Performs processing in response to the TCNT*nx* register interrupt.

[Syntax]

```
void    r_atuiv_timerknm_tcntnx_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the register number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_cmf*nx*_interrupt

Performs processing in response to the CMF*nx* register interrupt.

[Syntax]

```
void    r_atuiv_timerknm_cmfnx_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the register number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_callback_ocrc

Performs processing in response to the OCRC*nm* register interrupt.

Remark This API function is called as the callback routine of interrupt process r_atuiv_timerknm_interrupt corresponding to the OCRC*nm* register interrupt.

[Syntax]

```
void    r_atuiv_timer knm _callback_ocrc ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_callback_grc

Performs processing in response to the GRCnm register interrupt.

Remark This API function is called as the callback routine of interrupt process r_atuiv_timerknm_interrupt corresponding to the GRC*nm* register interrupt.

[Syntax]

```
void    r_atuiv_timer knm _callback_grc ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_underflow_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
void    r_atuiv_timerknm_underflow_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_compare*x*_interrupt

Performs processing in response to the compare match compare register and counter interrupt

[Syntax]

```
void    r_atuiv_timerknm_comparex_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the register number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_callback_overflow

Performs processing in response to the overflow interrupt.

Remark This API function is called as the callback routine of interrupt process r_atuiv_timerknm_interrupt corresponding to the overflow interrupt.

[Syntax]

```
void    r_atuiv_timer knm _callback_overflow_callback ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_callback_cycle

Performs processing in response to the cycle match interrupt.

Remark This API function is called as the callback routine of interrupt process r_atuiv_timerknm_interrupt corresponding to the cycle match interrupt.

[Syntax]

```
void    r_atuiv_timernm_channelm_cycle_callback ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_callback_duty

Performs processing in response to the duty match interrupt.

Remark This API function is called as the callback routine of interrupt process r_atuiv_timerknm_interrupt corresponding to the duty match interrupt.

[Syntax]

```
void    r_atuiv_timer knm _callback_duty ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_fifo_overflow_interrupt

Performs processing in response to the FIFO overflow interrupt.

[Syntax]

```
void    r_atuiv_timerknm_fifo_overflow_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_fifo_datafull_interrupt

Performs processing in response to the FIFO data full interrupt.

[Syntax]

```
void    r_atuiv_timerknm_fifo_datafull_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_atuiv_timer*knm*_tcnt*k*_overflow_interrupt

Performs processing in response to the counter overflow interrupt.

[Syntax]

```
void    r_atuiv_timer knm _tcnt k _overflow_interrupt ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*k*_OperationOn

Enables operation of each timer.

[Syntax]

```
void    R_ATUIV_Timerk_OperationOn ( void );
```

Remark *k* is the timer kind.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*k*_OperationOff

Disables operation of each timer.

[Syntax]

```
void    R_ATUIV_Timerk_OperationOff ( void );
```

Remark *k* is the timer kind.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Start

Starts each timer count.

[Syntax]

```
void    R_ATUIV_Timerknm_Start ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Stop

Ends each timer count.

[Syntax]

```
void    R_ATUIV_Timerknm_Stop ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```
void    R_ATUIV_Timerknm_Get_PulseWidth ( uint32_t *  width );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * width;`	Pointer to area in which to store the results of input pulse width

[Return value]

None.

R_ATUIV_Timer*knm*_Get_CaptureValue

Reads the input capture register of the timer.

[Syntax]

```
void    R_ATUIV_Timerknm_Get_CaptureValue ( uint32_t * value );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * value;`	Pointer to area in which to store the results of input capture register

[Return value]

None.

R_ATUIV_Timer*knm*_Set_Compare_Match

Sets the compare match register.

[Syntax]

```
void    R_ATUIV_Timerknm_Set_Compare_Match ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Set_One_Shot_Pulse

Renewal the one shot pulse register.

[Syntax]

```
void    R_ATUIV_Timerknm_Set_One_Shot_Pulse ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Forced_Compare_Match

Performs forced compare match.

[Syntax]

```
void    R_ATUIV_Timerknm_Forced_Compare_Match ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Forced_Output_Compare_Match

Performs forced output compare match.

[Syntax]

```
void    R_ATUIV_Timerknm_Forced_Output_Compare_Match ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Start_Down_Count

Starts the down count.

[Syntax]

```
void    R_ATUIV_Timerknm_Start_Down_Count ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Get_InputCapture*x*

Reads the input capture register of the timer.

[Syntax]

```
void    R_ATUIV_Timerknm_Get_InputCapturex ( uint32_t *  value );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * value;`	Pointer to area in which to store the results of input capture register

[Return value]

None.

R_ATUIV_Timer*knm*_*x*pin_Output_Normal

Output of pin provides normal output.

[Syntax]

```
void    R_ATUIV_Timerknm_xpin_Output_Normal ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_*x*pin_Output_Low

Output of pin provides low output.

[Syntax]

```
void    R_ATUIV_Timerknm_xpin_Output_Low ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, *x* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_xpin_Output_High

Output of pin provides high output.

[Syntax]

```
void    R_ATUIV_Timerknm_xpin_Output_High ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number, x is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_ATUIV_Timer*knm*_Get_Count

Reads the counter value of measurement.

[Syntax]

```
void    R_ATUIV_Timerknm_Get_Count ( uint16_t *  count );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint16_t * count;`	Pointer to area in which to store the counter value

[Return value]

None.

R_ATUIV_Timer*knm*_Get_PWM_Measure_Value

Reads the PWM wave of measurement.

[Syntax]

```
void    R_ATUIV_Timerknm_Get_PWM_Measure_Value ( uint32_t * low_width, uint32_t *
edge_width );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * low_width;`	Pointer to area in which to store the results of low level width
	`uint32_t * edge_width;`	Pointer to area in which to store the results of edge interval count

[Return value]

None.

R_ATUIV_Timer*knm*_Get_Measure_Value

Reads the value of measurement (number of edges, off-state duty cycle, PWM cycle, edge input time).

[Syntax]

```
void    R_ATUIV_Timerknm_Get_Measure_Value ( uint16_t * edge, uint32_t *
off_duty_cycle, uint32_t * pwm_cycle, uint32_t * edge_input_time );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint16_t * edge;`	Pointer to area in which to store the number of edges
O	`uint32_t * off_duty_cycle;`	Pointer to area in which to store the off-state duty cycle
O	`uint32_t * pwm_cycle;`	Pointer to area in which to store the PWM cycle
O	`uint32_t * edge_input_time;`	Pointer to area in which to store the edge input time

[Return value]

None.

R_ATUIV_Timer*knm*_Reset_FIFO

Resets FIFO to the idle state.

[Syntax]

```
void    R_ATUIV_Timerknm_Reset_FIFO ( void );
```

Remark *k* is the timer kind, *n* is the unit number, *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.14 Timer Array Unit B

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

Table 3.14 API Functions: [Timer array unit B]

API Function Name	Function
R_TAUBn_Create	Performs initialization necessary to control the timer array unit B*n*.
R_TAUBn_Create_UserInit	Performs user-defined initialization relating to the timer array unit B*n*.
r_taubn_channelm_interrupt	Performs processing in response to the timer interrupt.
R_TAUBn_Channelm_Start	Starts the count for channel *m*.
R_TAUBn_Channelm_Stop	Ends the count for channel *m*.
R_TAUBn_Channelm_Get_PulseWidth	Reads the input pulse width of the timer.

R_TAUB*n*_Create

Performs initialization necessary to control the timer array unit B*n*.

[Syntax]

```
void    R_TAUBn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAUB*n*_Create_UserInit

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

Remark This API functions is called as the R_TAUBn_Create callback routine.

[Syntax]

```
void    R_TAUBn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_taub*n*_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```
void    r_taubn_channelm_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUB*n*_Channel*m*_Start

Starts the count for channel *m*.

[Syntax]

```
void    R_TAUBn_Channelm_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUB*n*_Channel*m*_Stop

Ends the count for channel m.

[Syntax]

```
void    R_TAUBn_Channelm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUB*n*_Channel*m*_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```
void    R_TAUBn_Channelm_Get_PulseWidth ( uint32_t *  width );
```

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

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * width;`	Pointer to area in which to store the results of input pulse width

[Return value]

None.

3.2.15 Timer Array Unit D

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

Table 3.15 API Functions: [Timer array unit D]

API Function Name	Function
R_TAUDn_Create	Performs initialization necessary to control the timer array unit D*n*.
R_TAUDn_Create_UserInit	Performs user-defined initialization relating to the timer array unit D*n*.
r_taudn_channelm_interrupt	Performs processing in response to the timer interrupt.
R_TAUDn_Channelm_Start	Starts the count for channel *m*.
R_TAUDn_Channelm_Stop	Ends the count for channel *m*.
R_TAUDn_Channelm_Get_PulseWidth	Reads the input pulse width of the timer.

R_TAUD*n*_Create

Performs initialization necessary to control the timer array unit D*n*.

[Syntax]

```
void    R_TAUDn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAUD*n*_Create_UserInit

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

Remark This API functions is called as the R_TAUDn_Create callback routine.

[Syntax]

```
void    R_TAUDn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_taud*n*_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```
void    r_taudn_channelm_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUD*n*_Channel*m*_Start

Starts the count for channel *m*.

[Syntax]

```
void    R_TAUDn_Channelm_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUD*n*_Channel*m*_Stop

Ends the count for channel m.

[Syntax]

```
void    R_TAUDn_Channelm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUD*n*_Channel*m*_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```
void    R_TAUDn_Channelm_Get_PulseWidth ( uint32_t *  width );
```

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

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * width;`	Pointer to area in which to store the results of input pulse width

[Return value]

None.

3.2.16 Timer Array Unit J

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

Table 3.16 API Functions: [Timer array unit J]

API Function Name	Function
R_TAUJn_Create	Performs initialization necessary to control the timer array unit J*n*.
R_TAUJn_Create_UserInit	Performs user-defined initialization relating to the timer array unit J*n*.
r_taujn_channelm_interrupt	Performs processing in response to the timer interrupt.
R_TAUJn_Channelm_Start	Starts the count for channel *m*.
R_TAUJn_Channelm_Stop	Ends the count for channel *m*.
R_TAUJn_Channelm_Get_PulseWidth	Reads the input pulse width of the timer.

R_TAUJ*n*_Create

Performs initialization necessary to control the timer array unit J*n*.

[Syntax]

```
void    R_TAUJn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAUJ*n*_Create_UserInit

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

Remark This API functions is called as the R_TAUJn_Create callback routine.

[Syntax]

```
void    R_TAUJn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_tauj*n*_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```
void    r_tauj n _channel m _interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUJ*n*_Channel*m*_Start

Starts the count for channel *m*.

[Syntax]

```
void    R_TAUJn_Channelm_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUJ*n*_Channel*m*_Stop

Ends the count for channel m.

[Syntax]

```
void    R_TAUJn_Channelm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAUJ*n*_Channel*m*_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```
void    R_TAUJn_Channelm_Get_PulseWidth ( uint32_t *  width );
```

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

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * width;`	Pointer to area in which to store the results of input pulse width

[Return value]

None.

3.2.17 Timer Option

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

Table 3.17 API Functions: [Timer option]

API Function Name	Function
R_TAPAm_Create	Performs initialization necessary to control the timer option functions.
R_TAPAm_Create_UserInit	Performs user-defined initialization relating to the timer option functions.
R_TAPAm_Start	Enables asynchronous Hi-Z control.
R_TAPAm_Stop	Disables asynchronous Hi-Z control.
R_TAPAm_Trigger_Start	Sets the Hi-Z control signal to the low level.
R_TAPAm_Trigger_Stop	Sets the Hi-Z control signal to the high level.

R_TAPA*m*_Create

Performs initialization necessary to control the timer option functions.

[Syntax]

```
void    R_TAPAm_Create ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAPA*m*_Create_UserInit

Performs user-defined initialization relating to the timer option functions.

Remark This API functions is called as the R_TAPAm_Create callback routine.

[Syntax]

```
void    R_TAPAm_Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAPA*m*_Start

Enables asynchronous Hi-Z control.

[Syntax]

```
void    R_TAPAm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAPA*m*_Stop

Disables asynchronous Hi-Z control.

[Syntax]

```
void    R_TAPAm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAPA*m*_Trigger_Start

Sets the Hi-Z control signal to the low level.

[Syntax]

```
void    R_TAPAm_Trigger_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAPA*m*_Trigger_Stop

Sets the Hi-Z control signal to the high level.

[Syntax]

```
void    R_TAPAm_Trigger_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.18 Peripheral Interconnection

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

Table 3.18 API Functions: [Peripheral interconnection]

API Function Name	Function
R_PICn_Create	Performs initialization necessary to control the peripheral interconnection functions.
R_PICn_Create_UserInit	Performs user-defined initialization relating to the peripheral interconnection functions.
R_PICn_Timer_SyncStart	Generates start triggers of the timers for which simultaneous start is enabled.

R_PIC*n*_Create

Performs initialization necessary to control the peripheral interconnection functions.

[Syntax]

```
void    R_PICn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_PIC*n*_Create_UserInit

Performs user-defined initialization relating to the peripheral interconnection functions.

Remark This API functions is called as the R_PICn_Create callback routine.

[Syntax]

```
void    R_PICn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_PIC*n*_Timer_SyncStart

Generates start triggers of the timers for which simultaneous start is enabled.

[Syntax]

```
void    R_PICn_Timer_SyncStart ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.19 A/D converter

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

Table 3.19 API Functions: [A/D converter]

API Function Name	Function
R_ADCn_Create	Performs initialization necessary to control the A/D converter functions.
R_ADCn_Create_UserInit	Performs user-defined initialization relating to the A/D converter functions.
r_adcn_error_interrupt	Performs processing in response to the A/D error interrupt.
r_adcn_scan_groupm_end_interrupt	Performs processing in response to the scan group end interrupt.
r_adcn_multiplexer_request_interrupt	Performs processing in response to the MPX interrupt.
r_adcn_adc_summation_channelm_end_interrupt	Performs processing in response to the accumulation end interrupt.
R_ADCn_Halt	Halts A/D converter.
R_ADCn_SetMultiplexerCommand	Sets MPX command.
R_ADCn_ScanGroupm_Start	Starts ADC scan group conversion.
R_ADCn_ScanGroupm_GetResult	Reads the results of ADC scan group conversion.
R_ADCn_ScanGroupm_GetFloatingPointDataResult	Reads the results of ADC scan group conversion (converted to the floating-point format).
R_ADCn_ScanGroupm_TimerStart	Starts ADC scan group timer.
R_ADCn_ScanGroupm_TimerStop	Ends ADC scan group timer.
R_ADCn_ADCSummation_Channelm_GetResult	Reads the results of accumulation data.
R_ADCn_ADCSummation_Start	Starts summation function.
R_ADCn_ADCSummation_Stop	Ends summation function.
R_ADC_SyncStart	Starts ADC scan group conversion set as enable scan group synchronization start.
R_ADC_SyncTimerStart	Starts ADC scan group conversion set as enable A/D timer synchronization start.
R_ADCn_ScanGroupm_OperationOn	Starts ADC scan group scan.
R_ADCn_TH_Groupx_Start	Starts ADC T&H group hold.
R_ADCn_TH_Sampling_Start	Starts ADC T&H sampling.

R_ADC*n*_Create

Performs initialization necessary to control the A/D converter functions.

[Syntax]

```
void    R_ADCn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_Create_UserInit

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

Remark This API functions is called as the R_ADCn_Create callback routine.

[Syntax]

```
void    R_ADCn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_adc*n*_error_interrupt

Performs processing in response to the A/D error interrupt.

[Syntax]

```
void    r_adcn_error_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_adc*n*_scan_group*m*_end_interrupt

Performs processing in response to the scan group end interrupt.

[Syntax]

```
void    r_adcn_scan_groupm_end_interrupt ( void );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

None.

[Return value]

None.

r_adc*n*_multiplexer_request_interrupt

Performs processing in response to the MPX interrupt.

[Syntax]

```
void    r_adcn_multiplexer_request_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_adc*n*_adc_summation_channel*m*_end_interrupt

Performs processing in response to the accumulation end interrupt.

[Syntax]

```
void    r_adcn_adc_summation_channelm_end_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_Halt

Halts A/D converter.

[Syntax]

```
void    R_ADCn_Halt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_SetMultiplexerCommand

Sets MPX command.

[Syntax]

```
void    R_ADCn_SetMultiplexerCommand ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ScanGroup*m*_Start

Starts ADC scan group conversion.

[Syntax]

```
void    R_ADCn_ScanGroupm_Start ( void );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ScanGroup*m*_GetResult

Reads the results of ADC scan group conversion.

[Syntax]

```
void    R_ADCn_ScanGroupm_GetResult ( uint16_t * buffer );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

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

[Return value]

None.

R_ADC*n*_ScanGroup*m*_GetFloatingPointDataResult

Reads the results of ADC scan group conversion (converted to the floating-point format).

[Syntax]

```
void    R_ADCn_ScanGroupm_GetFloatingPointDataResult ( uint32_t * buffer );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * buffer;`	Pointer to area in which to store the results of A/D conversion

[Return value]

None.

R_ADC*n*_ScanGroup*m*_TimerStart

Starts ADC scan group timer.

[Syntax]

```
void    R_ADCn_ScanGroupm_TimerStart ( void );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ScanGroup*m*_TimerStop

Ends ADC scan group timer.

[Syntax]

```
void    R_ADCn_ScanGroupm_TimerStop ( void );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ADCSummation_Cannel*m*_GetResult

Reads the results of accumulation data.

[Syntax]

```
void    R_ADCn_ADCSummation_Channelm_GetResult ( uint32_t * buffer );
```

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

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * buffer;`	Pointer to area in which to store the results of accumulation data

[Return value]

None.

R_ADC*n*_ADCSummation_Start

Starts summation function.

[Syntax]

```
void    R_ADCn_ADCSummation_Start ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ADCSummation_Stop

Ends summation function.

[Syntax]

```
void    R_ADCn_ADCSummation_Stop ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ADC_SyncStart

Starts ADC scan group conversion set as enable scan group synchronization start.

[Syntax]

```
void    R_ADC_SyncStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_SyncTimerStart

Starts ADC scan group conversion set as enable A/D timer synchronization start.

[Syntax]

```
void    R_ADC_SyncTimerStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_ScanGroup*m*_OperationOn

Starts ADC scan group scan.

[Syntax]

```
void    R_ADCn_ScanGroupm_OperationOn ( void );
```

Remark *n* is the unit number, *m* is the scan group number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_TH_Group*x*_Start

Starts ADC T&H group hold.

[Syntax]

```
void    R_ADCn_TH_Groupx_Start ( void );
```

Remark *n* is the unit number, *x* is the group number.

[Argument(s)]

None.

[Return value]

None.

R_ADC*n*_TH_Sampling_Start

Starts ADC T&H sampling.

[Syntax]

```
void    R_ADCn_TH_Sampling_Start ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.20 Delta-Sigma AD converter

Below is a list of API functions output by the Code Generator for Delta-Sigma AD converter use.

Table 3.20 API Functions: [Delta-Sigma AD converter]

API Function Name	Function
R_DSADC_Create	Performs initialization necessary to control the ΔΣ A/D converter functions.
R_DSADC_Create_UserInit	Performs user-defined initialization relating to the ΔΣ A/D converter functions.
r_dsadc_error_interrupt	Performs processing in response to the ΔΣ A/D error interrupt.
R_DSADC_SyncStart	Starts ΔΣ AD conversion set as enable AD synchronization start.
R_DSADCm_Start	Starts ΔΣ A/D conversion.
R_DSADCm_Stop	Ends ΔΣ A/D conversion.
R_DSADCm_GetResult	Reads the results of ΔΣ A/D conversion.

R_DSADC_Create

Performs initialization necessary to control the ΔΣ A/D converter functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_dsadc_error_interrupt

Performs processing in response to the ΔΣ A/D error interrupt.

[Syntax]

```
void    r_dsadc_error_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_SyncStart

Starts ΔΣ AD conversion set as enable AD synchronization start.

[Syntax]

```
void    R_DSADC_SyncStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC*m*_Start

Starts ΔΣ A/D conversion.

[Syntax]

```
void    R_DSADCm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DSADC*m*_Stop

Ends ΔΣ A/D conversion.

[Syntax]

```
void    R_DSADCm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DSADC*m*_GetResult

Reads the results of ΔΣ A/D conversion.

[Syntax]

```
void    R_DSADCm_GetResult ( uint32_t * result );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * result;`	Pointer to area in which to store the results of A/D conversion

[Return value]

None.

3.2.21 Digital Filter

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

Table 3.21 API Functions: [Digital filter]

API Function Name	Function
R_DFE_Create	Performs initialization necessary to control the digital filter functions.
R_DFE_Create_UserInit	Performs user-defined initialization relating to the digital filter functions.
r_dfe_error_interrupt	Performs processing in response to the DFE error interrupt.
R_DFE_Set_SoftwareData	Sets the data to be processed.in a target filter.
R_DFE_Generate_SoftwareTrigger	Generates the software trigger.
R_DFE_Channelm_Create	Performs initialization necessary to control the digital filter channel functions.
R_DFE_Channelm_Create_UserInit	Performs user-defined initialization relating to the digital filter channel functions.
r_dfe_channelm_interrupt	Performs processing in response to the DFE channel interrupt.
r_dfe_channelm_callback_output_data	Performs processing in response to the output data interrupt.
r_dfe_channelm_callback_condition_match	Performs processing in response to the condition match interrupt.
R_DFE_Channelm_Enable	Enables the DFE channel.
R_DFE_Channelm_Disable	Disables the DFE channel.
R_DFE_Channelm_GetResult	Reads the results of DFE processing calculation.

R_DFE_Create

Performs initialization necessary to control the digital filter functions.

[Syntax]

```
void    R_DFE_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DFE_Create_UserInit

Performs user-defined initialization relating to the digital filter functions.

Remark This API functions is called as the R_DFE_Create callback routine.

[Syntax]

```
void    R_DFE_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dfe_error_interrupt

Performs processing in response to the DFE error interrupt.

[Syntax]

```
void    r_dfe_error_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DFE_Set_SoftwareData

Sets the data to be processed.in a target filter.

[Syntax]

```
void    R_DFE_Set_SoftwareData ( uint8_t tag, int16_t const data );
```

[Argument(s)]

I/O	Argument(s)	Description
I	`uint8_t tag;`	Set the same value as the channel tag
I	`int16_t const data;`	Software input data

[Return value]

None.

R_DFE_Generate_SoftwareTrigger

Generates the software trigger.

[Syntax]

```
void    R_DFE_Generate_SoftwareTrigger ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DFE_Channel*m*_Create

Performs initialization necessary to control the digital filter channel functions.

[Syntax]

```
void   R_DFE_Channelm_Create (void);
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DFE_Channel*m*_Create_UserInit

Performs user-defined initialization relating to the digital filter channel functions.

Remark This API functions is called as the R_DFE_Channelm_Create callback routine.

[Syntax]

```
void    R_DFE_Channel m _Create_UserInit ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dfe_channel*m*_interrupt

Performs processing in response to the DFE channel interrupt.

[Syntax]

```
void    r_dfe_channelm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Performs processing in response to the output data interrupt.

r_dfe_channel*m*_callback_output_data

Performs processing in response to the output data interrupt.

Remark This API functions is called as the r_dfe_channelm_interrupt callback routine.

[Syntax]

```
void   r_dfe_channelm_callback_output_data (void);
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dfe_channel*m*_callback_condition_match

Performs processing in response to the condition match interrupt.

Remark This API functions is called as the r_dfe_channelm_interrupt callback routine.

[Syntax]

```
void    r_dfe_channelm_callback_condition_match ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DFE_Channel*m*_Enable

Enables the DFE channel.

[Syntax]

```
void    R_DFE_Channelm_Enable ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DFE_Channel*m*_Disable

Disables the DFE channel.

[Syntax]

```
void    R_DFE_Channelm_Disable ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DFE_Channel*m*_GetResult

Reads the results of DFE processing calculation.

[Syntax]

```
void    R_DFE_Channelm_GetResult ( int32_t * buffer );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	`int32_t * buffer;`	Pointer to area in which to store the results of DFE processing calculation

[Return value]

None.

3.2.22 Data CRC

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

Table 3.22 API Functions: [Data CRC]

API Function Name	Function
R_DCRAn_Create	Performs initialization necessary to control the data CRC functions.
R_DCRAn_Create_UserInit	Performs user-defined initialization relating to the data CRC functions.
R_DCRAn_Input32bitData	Sets the calculation data for 32 bit width.
R_DCRAn_Input16bitData	Sets the calculation data for 16 bit width.
R_DCRAn_Input8bitData	Sets the calculation data for 8 bit width.
R_DCRAn_GetResult_32bitData	Reads the results of CRC calculation for 32 bit width.
R_DCRAn_GetResult_16bitData	Reads the results of CRC calculation for 16 bit width.

R_DCRA*n*_Create

Performs initialization necessary to control the data CRC functions.

[Syntax]

```
void    R_DCRAn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DCRA*n*_Create_UserInit

Performs user-defined initialization relating to the data CRC functions.

Remark This API functions is called as the R_DCRAn_Create callback routine.

[Syntax]

```
void    R_DCRAn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DCRA*n*_Input32bitData

Sets the calculation data for 32 bit width.

[Syntax]

```
void    R_DCRAn_Input32bitData ( const uint32_t * data, uint32_t data_num );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint32_t * data;`	Pointer to a buffer storing the calculation data
I	`uint32_t data_num;`	Total amount of calculation data

[Return value]

None.

R_DCRA*n*_Input16bitData

Sets the calculation data for 16 bit width.

[Syntax]

```
void    R_DCRAn_Input16bitData ( const uint16_t * data, uint32_t data_num );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint16_t * data;`	Pointer to a buffer storing the calculation data
I	`uint32_t data_num;`	Total amount of calculation data

[Return value]

None.

R_DCRA*n*_Input8bitData

Sets the calculation data for 8 bit width.

[Syntax]

```
void    R_DCRAn_Input32bitData ( const uint8_t * data, uint32_t data_num );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	`const uint8_t * data;`	Pointer to a buffer storing the calculation data
I	`uint32_t data_num;`	Total amount of calculation data

[Return value]

None.

R_DCRA*n*_GetResult_32bitData

Reads the results of CRC calculation for 32 bit width.

[Syntax]

```
void    R_DCRAn_GetResult_32bitData ( uint32_t * data );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint32_t * data;`	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_DCRA*n*_GetResult_16bitData

Reads the results of CRC calculation for 16 bit width.

[Syntax]

```
void    R_DCRAn_GetResult_16bitData ( uint16_t * data );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	`uint16_t * data;`	Pointer to area in which to store the results of calculation data

[Return value]

None.

3.2.23 Real-Time Clock

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

Table 3.23 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_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_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_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_Set_ConstPeriodInterruptOn	Sets the cycle of the periodic interrupts, then starts the periodic interrupt function.
R_RTC_Set_ConstPeriodInterruptOff	Ends the periodic interrupt function.
R_RTC_Set_1secondInterruptOn	Starts the 1 second interrupt function.
R_RTC_Set_1secondInterruptOff	Ends the 1 second interrupt function.
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_interrupt_periodic	Performs processing in response to the periodic interrupt.
r_rtc_interrupt_alarm	Performs processing in response to the alarm interrupt.
r_rtc_interrupt_1second	Performs processing in response to the 1 second interrupt.

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_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_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_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 );
```

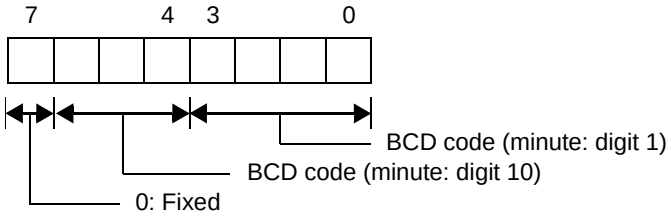
[Argument(s)]

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

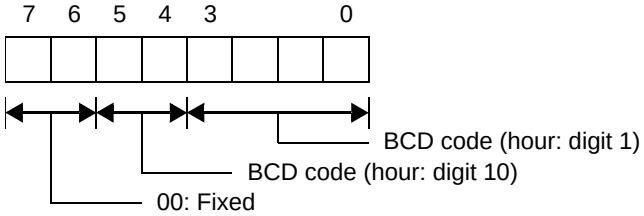
Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).

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

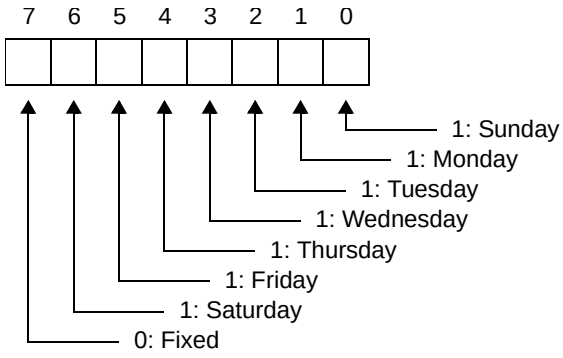
- alarmwm (Minute)
  Below are shown the meanings of each bit of the structure member alarmwm.



- 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_Set_ConstPeriodInterruptOn

Sets the cycle of the periodic interrupts, then starts the periodic 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 QUARTERSEC: 0.25 seconds 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 periodic interrupt function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_1secondInterruptOn

Starts the 1 second interrupt function.

[Syntax]

```
void    R_RTC_Set_1secondInterruptOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_1secondInterruptOff

Ends the 1 second interrupt function.

[Syntax]

```
void    R_RTC_Set_1secondInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOn

Enables output of the correction clock (1 Hz).

[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).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_rtc_interrupt_periodic

Performs processing in response to the periodic interrupt.

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

[Syntax]

```
void    r_rtc_interrupt_periodic ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_interrupt_alarm

Performs processing in response to the alarm interrupt.

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

[Syntax]

```
void    r_rtc_interrupt_alarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_interrupt_1second

Performs processing in response to the 1 second interrupt.

Remark This API function is called as the interrupt process corresponding to the 1 second interrupt.

[Syntax]

```
void    r_rtc_interrupt_1second ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.24 Key Return

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

Table 3.24 API Functions: [Key Return]

API Function Name	Function
R_KEY_Create	Performs initialization necessary to control the key return functions.
R_KEY_Create_UserInit	Performs user-defined initialization relating to the key return functions.
r_key_interrupt	Performs processing in response to the key return interrupt.
R_KEY_Start	Enables the acceptance of the key return interrupt.
R_KEY_Stop	Disables the acceptance of the key return interrupt.

R_KEY_Create

Performs initialization necessary to control the key return functions.

[Syntax]

```
void    R_KEY_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create_UserInit

Performs user-defined initialization relating to the key return functions.

Remark This API functions is called as theR_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 return interrupt.

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

[Syntax]

```
void    r_key_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Start

Enables the acceptance of the key return interrupt.

[Syntax]

```
void    R_KEY_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Stop

Disables the acceptance of the key return interrupt.

[Syntax]

```
void    R_KEY_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.25 Stand-By Controller

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

Table 3.25 API Functions: [Stand-By Controller]

API Function Name	Function
R_STBC_Start_Stop_Mode	Start stand-by (STOP mode).
R_STBC_Prepare_Stop_Mode	Performs user-defined processing relating to the preparation to start stand-by (STOP mode).
R_STBC_Start_Deep_Stop_Mode	Start stand-by (DeepSTOP mode).
R_STBC_Prepare_Deep_Stop_Mode	Performs user-defined processing relating to the preparation to start stand-by (DeepSTOP mode).
R_STBC_Deep_Stop_Loop	Performs wait processing of stand-by (DeepSTOP mode).
R_STBC_Prepare_Stop_Mode_Set_Peripheral	Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (STOP mode).
R_STBC_Prepare_Stop_Mode_Set_Interrupt	Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (STOP mode).
R_STBC_Prepare_Stop_Mode_Set_Clock_Mask	Performs user-defined processing relating to the preparation (set the clock stop mask register) to start stand-by (STOP mode).
R_STBC_Prepare_Stop_Mode_Set_Clock_Source	Performs user-defined processing relating to the preparation (oscillate or stop each clock source) to start stand-by (STOP mode).
R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral	Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (DeepSTOP mode).
R_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt	Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (DeepSTOP mode).

R_STBC_Start_Stop_Mode

Start stand-by (STOP mode).

Remark R_STBC_Prepare_Stop_Mode must be called before this API function is called.

[Syntax]

```
void    R_STBC_Start_Stop_Mode ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Stop_Mode

Performs user-defined processing relating to the preparation to start stand-by (STOP mode).

[Syntax]

```
void    R_STBC_Prepare_Stop_Mode ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Start_Deep_Stop_Mode

Start stand-by (DeepSTOP mode).

Remark R_STBC_Prepare_Deep_Stop_Mode must be called before this API function is called.

[Syntax]

```
void    R_STBC_Start_Deep_Stop_Mode ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Deep_Stop_Mode

Performs user-defined processing relating to the preparation to start stand-by (DeepSTOP mode).

[Syntax]

```
void    R_KEY_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Deep_Stop_Loop

Performs wait processing of stand-by (DeepSTOP mode).

[Syntax]

```
void    R_STBC_Deep_Stop_Loop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Stop_Mode_Set_Peripheral

Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (STOP mode).

[Syntax]

```
void    R_STBC_Prepare_Stop_Mode_Peripheral ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Stop_Mode_Set_Interrupt

Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (STOP mode).

[Syntax]

```
void    R_STBC_Prepare_Stop_Mode_Set_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Stop_Mode_Set_Clock_Mask

Performs user-defined processing relating to the preparation (set the clock stop mask register) to start stand-by (STOP mode).

[Syntax]

```
void    R_STBC_Prepare_Stop_Mode_Set_Clock_Mask ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Stop_Mode_Set_Clock_Source

Performs user-defined processing relating to the preparation (oscillate or stop each clock source) to start stand-by (STOP mode).

[Syntax]

```
void    R_STBC_Prepare_Stop_Mode_Set_Clock_Source ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral

Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (DeepSTOP mode).

[Syntax]

```
void    R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt

Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (DeepSTOP mode).

[Syntax]

```
void    R_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 01, 2015	-	First Edition issued
1.01	Mar 01, 2016	38 - 40	3.2.4 Interrupt API addition
		71 - 83	3.2.7 Clock Serial Interface G Chapter addition
		120 - 132	3.2.10 UART Interface Chapter addition
		186 - 192	3.2.14 Timer Array Unit B Chapter addition
		193 - 199	3.2.15 Timer Array Unit D Chapter addition
		200 - 206	3.2.16 Timer Array Unit J Chapter addition
		237 - 239	3.2.19 A/D converter API addition
		270 - 291	3.2.23 Real-Time Clock Chapter addition
		292 - 297	3.2.24 Key Return Chapter addition
		298 - 309	3.2.25 Stand-By Controller Chapter addition

CS+ Code Generator Tool User's Manual:
RH850 API Reference

Publication Date: Rev.1.01 Mar 01, 2016

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

California Eastern Laboratories, Inc.
4590 Patrick Henry Drive, Santa Clara, California 95054-1817, U.S.A.
Tel: +1-408-919-2500, Fax: +1-408-988-0279

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

CS+ Code Generator Tool

Renesas Electronics Corporation

R20UT3567EJ0101