User's Manual

RENESAS

AP4

Integrated Development Environment

User's Manual: RZ/T1 API Reference

Target Device

RZ Family

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

Renesas Electronics
www.renesas.com

Rev.1.02 Mar 13, 2020

How to Use This Manual

Readers	The target readers of this manual are the application system engineers who use the Code Generator and need to understand its function.
Purpose	The purpose of this manual is to explain the user for understanding and using the Code Generator functions. We aim to help their system development including their hardware and software.
Organization	This manual can be broadly divided into the following units. 1.GENERAL 2.OUTPUT FILES 3.API FUNCTIONS
How to Read This Manual	It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{\text{XXX}}$ (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

The Code Generator is a software tool that automatically generates device drivers.
This chapter gives an overview of the design tool.

1.1 Overview

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

1.2 Features

The Code Generator tool has the following features.

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

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

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

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

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

2. OUTPUT FILES

This appendix describes the files output by the Code Generator.

2.1 Description

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

Table 2.1 Output File List

Peripheral Function	File Name	API Function Name
Common	r_cg_main.c	main R_MAIN_UserInit
	r_cg_mpc.c	R_MPC_Create R_MPC_Create_UserInit
	r_cg_systeminit.c	R_Systeminit
	r_cg_intprg.c	r_set_exception_handler r_fiq_handler
	r_cg_macrodriver.h	—
	r_cg_userdefine.h	—
	r_cg_interrupthandlers.h	—
	r_cg_mpc.h	—
	r_cg_nestintr_wrap.asm	—
Clock generator	r_cg_cgc.c	R_CGC_Create
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_ostde_interrupt
	r_cg_cgc.h	—
Interrupt Controller	r_cg_icu.c	R_ICU_Create R_ICU_IRQn_Start R_ICU_IRQn_Stop R_ICU_ETHPHYIn_Start R_ICU_ETHPHYIn_Stop
	r_cg_icu_user.c	R_ICU_Create_UserInit r_icu_nmi_interrupt r_icu_irqn_interrupt r_icu_ethphyin_interrupt
	r_cg_icu.h	—
Bus State Controller	r_cg_bsc.c	R_BSC_Create R_BSC_InitializeSDRAM R_BSC_SDRAMPowerDown_Start R_BSC_SDRAMPowerDown_Stop R_BSC_SDRAMDeepPowerDown_Start R_BSC_SDRAMDeepPowerDown_Stop
	r_cg_bsc_user.c	R_BSC_Create_UserInit r_bsc_bsccmi_interrupt r_bsc_tostf_interrupt
	r_cg_bsc.h	—

Peripheral Function	File Name	API Function Name
DMA Controller	r_cg_dmac.c	R_DMACn_Create R_DMACn_Set_SoftwareTrigger R_DMACm_Cn_Start R_DMACm_Cn_Stop R_DMACm_Cn_Suspend R_DMACm_Cn_SuspendClear
	r_cg_dmac_user.c	R_DMACn_Create_UserInit r_dmac0_dmaintn_interrupt r_dmac1_dmaint2_interrupt r_dmac0_dmasrq0_interrupt r_dmac1_dmasrq1_interrupt r_dmac0_dmaerr0_interrupt r_dmac1_dmaerr1_interrupt r_callback_dmac0_dmaintn_interrupt r_callback_dmac1_dmaint2_interrupt
	r_cg_dmac.h	—
Event Link Controller	r_cg_elc.c	R_ELC_Create R_ELC_Start R_ELC_Stop R_ELC_GenerateSoftwareEvent R_ELC_Get_PortBuffern R_ELC_Set_PortBuffern
	r_cg_elc_user.c	R_ELC_Create_UserInit r_elc_elcirqn_interrupt
	r_cg_elc.h	—
I/O Ports	r_cg_port.c	R_PORT_Create
	r_cg_port_user.c	R_PORT_Create_UserInit
	r_cg_port.h	—
Multi-Function Timer Pulse Unit	r_cg_mtu3.c	R_MTU3_Create R_MTU3_Cm_Start R_MTU3_Cm_Stop
	r_cg_mtu3_user.c	R_MTU3_Create_UserInit r_mtu3_tgiam_interrupt r_mtu3_tgibm_interrupt r_mtu3_tgicm_interrupt r_mtu3_tgidm_interrupt r_mtu3_tgie0_interrupt r_mtu3_tgif0_interrupt r_mtu3_tcivm_interrupt r_mtu3_tcium_interrupt r_mtu3_tgiu5_interrupt r_mtu3_tgiv5_interrupt r_mtu3_tgiw5_interrupt r_mtu3_c4_tgia4_interrupt r_mtu3_c4_tgib4_interrupt r_mtu3_c4_tciv4_interrupt r_mtu3_c7_tgia7_interrupt r_mtu3_c7_tgib7_interrupt r_mtu3_c7_tciv7_interrupt
	r_cg_mtu3.h	—

Peripheral Function	File Name	API Function Name
Port Output Enable 3	r_cg_poe3.c	R_POE3_Create R_POE3_Start R_POE3_Stop R_POE3_Set_HiZ_MTUm R_POE3_Clear_HiZ_MTUm R_POE3_Set_HiZ_GPT3 R_POE3_Clear_HiZ_GPT3
	r_cg_poe3_user.c	R_POE3_Create_UserInit r_poe3_oein_interrupt
	r_cg_poe3.h	—
General PWM Timer	r_cg_gpt.c	R_GPT_Create R_GPTn_Start R_GPTn_Stop R_GPTn_HardwareStart R_GPTn_HardwareStop
	r_cg_gpt_user.c	R_GPT_Create_UserInit r_gpt_etgin_interrupt r_gpt_etgip_interrupt r_gpt_gtcian_interrupt r_gpt_gtcibn_interrupt r_gpt_gtcicn_interrupt r_gpt_gtcidn_interrupt r_gpt_gtcien_interrupt r_gpt_gtcifn_interrupt r_gpt_gdten_interrupt r_gpt_gtcivn_interrupt r_gpt_gtciun_interrupt
	r_cg_gpt.h	—
16-Bit Timer Pulse Unit	r_cg_tpu.c	R_TPU_Create R_TPUn_Start R_TPUn_Stop
	r_cg_tpu_user.c	R_TPU_Create_UserInit r_tpu_tgina_interrupt r_tpu_tginb_interrupt r_tpu_tginc_interrupt r_tpu_tgind_interrupt r_tpu_tcinv_interrupt r_tpu_tcinu_interrupt
	r_cg_tpu.h	—
Programmable Pulse Generator	r_cg_ppg.c	R_PPG_Create
	r_cg_ppg_user.c	R_PPG_Create_UserInit
	r_cg_ppg.h	—
Compare Match Timer	r_cg_cmt.c	R_CMTn_Create R_CMTn_Start R_CMTn_Stop
	r_cg_cmt_user.c	R_CMTn_Create_UserInit r_cmt_cmin_interrupt
	r_cg_cmt.h	—

Peripheral Function	File Name	API Function Name
Compare Match Timer W	r_cg_cmtw.c	R_CMTWm_Create R_CMTWm_Start R_CMTWm_Stop
	r_cg_cmtw_user.c	R_CMTWm_Create_UserInit r_cmtw_cmwim_interrupt r_cmtw_icnim_interrupt r_cmtw_ocnim_interrupt
	r_cg_cmtw.h	—
Watchdog Timer	r_cg_wdt.c	R_WDTn_Create R_WDTn_Restart
	r_cg_wdt_user.c	R_WDTn_Create_UserInit r_wdtn_undff_refef_interrupt
	r_cg_wdt.h	—
Independent Watchdog Timer	r_cg_iwdt.c	R_IWDT_Create R_IWDT_Restart
	r_cg_iwdt_user.c	R_IWDT_Create_UserInit r_iwdt_undff_refef_interrupt
	r_cg_iwdt.h	—
Serial Communications Interface with FIFO	r_cg_scifa.c	R_SCIFAn_Create R_SCIFAn_Start R_SCIFAn_Stop R_SCIFAn_Serial_Send R_SCIFAn_Serial_Receive R_SCIFAn_Serial_Send_Receive
	r_cg_scifa_user.c	R_SCIFAn_Create_UserInit r_scifan_txifn_interrupt r_scifan_rxifn_interrupt r_scifan_brifn_interrupt r_scifan_drifn_interrupt r_scifan_teifn_interrupt r_scifan_erifn_interrupt r_scifan_callback_transmitend r_scifan_callback_receiveend r_scifan_callback_error
	r_cg_scifa.h	—

Peripheral Function	File Name	API Function Name
I²C Bus Interface	r_cg_riic.c	R_RIICn_Create R_RIICn_Start R_RIICn_Stop R_RIICn_Master_Send R_RIICn_Master_Send_Without_Stop R_RIICn_Master_Receive R_RIICn_Slave_Send R_RIICn_Slave_Receive R_RIICn_StartCondition R_RIICn_StopCondition
	r_cg_riic_user.c	R_RIICn_Create_UserInit r_riicn_error_interrupt r_riicn_receive_interrupt r_riicn_transmit_interrupt r_riicn_transmitend_interrupt r_riicn_callback_receiveerror r_riicn_callback_transmitend r_riicn_callback_receiveend
	r_cg_riic.h	—
Serial Peripheral Interface	r_cg_rspi.c	R_RSPIn_Create R_RSPIn_Start R_RSPIn_Stop R_RSPIn_Send R_RSPIn_Send_Receive
	r_cg_rspi_user.c	R_RSPIn_Create_UserInit r_rspin_receive_interrupt r_rspin_transmit_interrupt r_rspin_error_interrupt r_rspin_idle_interrupt r_rspin_callback_receiveend r_rspin_callback_error r_rspin_callback_transmitend
	r_cg_rspi.h	—
SPI Multi I/O Bus Controller	r_cg_spibsc.c	R_SPIBSC_Create R_SPIBSC_EAVUpperAddressChange R_SPIBSC_SPIRead R_SPIBSC_SPIWrite R_SPIBSC_SPIRead_Write
	r_cg_spibsc_user.c	R_SPIBSC_Create_UserInit
	r_cg_spibsc.h	—
CRC Operation Units	r_cg_crc.c	R_CRC_SetCRC8_2F R_CRC_SetCRC8_SAE R_CRC_SetCRC16_CCITT R_CRC_SetCRC32_ETHER R_CRC_Input_Data R_CRC_Get_Result
	r_cg_crc.h	—

Peripheral Function	File Name	API Function Name
ΔΣ Interface	r_cg_dsmif.c	R_DSMIF_Create R_DSMIF_UVW_Start R_DSMIF_UVW_Stop R_DSMIF_X_Start R_DSMIF_X_Stop
	r_cg_dsmif_user.c	R_DSMIF_Create_UserInit
	r_cg_dsmif.h	—
Error Control Module	r_cg_ecm.c	R_ECM_Create R_ECM_Pseudo_WDT0_Error_Start R_ECM_Pseudo_WDT0_Error_Stop R_ECM_Pseudo_IWDTa_Error_Start R_ECM_Pseudo_IWDTa_Error_Stop R_ECM_Pseudo_CGC_Error_Start R_ECM_Pseudo_CGC_Error_Stop R_ECM_Pseudo_ADC_Unit0_Error_Start R_ECM_Pseudo_ADC_Unit0_Error_Stop R_ECM_Pseudo_ADC_Unit1_Error_Start R_ECM_Pseudo_ADC_Unit1_Error_Stop R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Start R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop R_ECM_Pseudo_DOC_Error_Start R_ECM_Pseudo_DOC_Error_Stop R_ECM_Pseudo_BSC_Error_Start R_ECM_Pseudo_BSC_Error_Stop R_ECM_Pseudo_Error35_Error_Start R_ECM_Pseudo_Error35_Error_Stop R_ECM_Pseudo_Error36_Error_Start R_ECM_Pseudo_Error36_Error_Stop R_ECM_Pseudo_Error37_Error_Start R_ECM_Pseudo_Error37_Error_Stop R_ECM_Pseudo_Error38_Error_Start R_ECM_Pseudo_Error38_Error_Stop R_ECM_Pseudo_Error39_Error_Start R_ECM_Pseudo_Error39_Error_Stop R_ECM_Pseudo_Error40_Error_Start R_ECM_Pseudo_Error40_Error_Stop R_ECM_Pseudo_Error41_Error_Start R_ECM_Pseudo_Error41_Error_Stop R_ECM_Pseudo_ECM_CompareError_Error_Start R_ECM_Pseudo_ECM_CompareError_Error_Stop R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop
	r_cg_ECM_user.c	R_ECM_Create_UserInit r_ecm_nmi_interrupt r_ecm_errd_interrupt r_ecm_compareerror_interrupt
	r_cg_ecm.h	—

Peripheral Function	File Name	API Function Name
12-Bit A/D Converter	r_cg_s12ad.c	R_S12ADn_Create R_S12ADn_Start R_S12ADn_Stop R_S12ADn_Get_ValueResult R_S12ADn_Set_CompareValue
	r_cg_s12ad_user.c	R_S12ADn_Create_UserInit r_s12ad_s12adn_interrupt r_s12ad_s12gbadin_interrupt r_s12ad_s12cmpn_interrupt r_s12ad_s12aden_interrupt
	r_cg_s12ad.h	—
Data Operation Circuit	r_cg_doc.c	R_DOC_Create R_DOC_SetMode R_DOC_WriteData R_DOC_GetResult R_DOC_ClearFlag
	r_cg_doc_user.c	R_DOC_Create_UserInit r_doc_dopcf_interrupt
	r_cg_doc.h	—

3. API FUNCTIONS

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

3.1 Overview

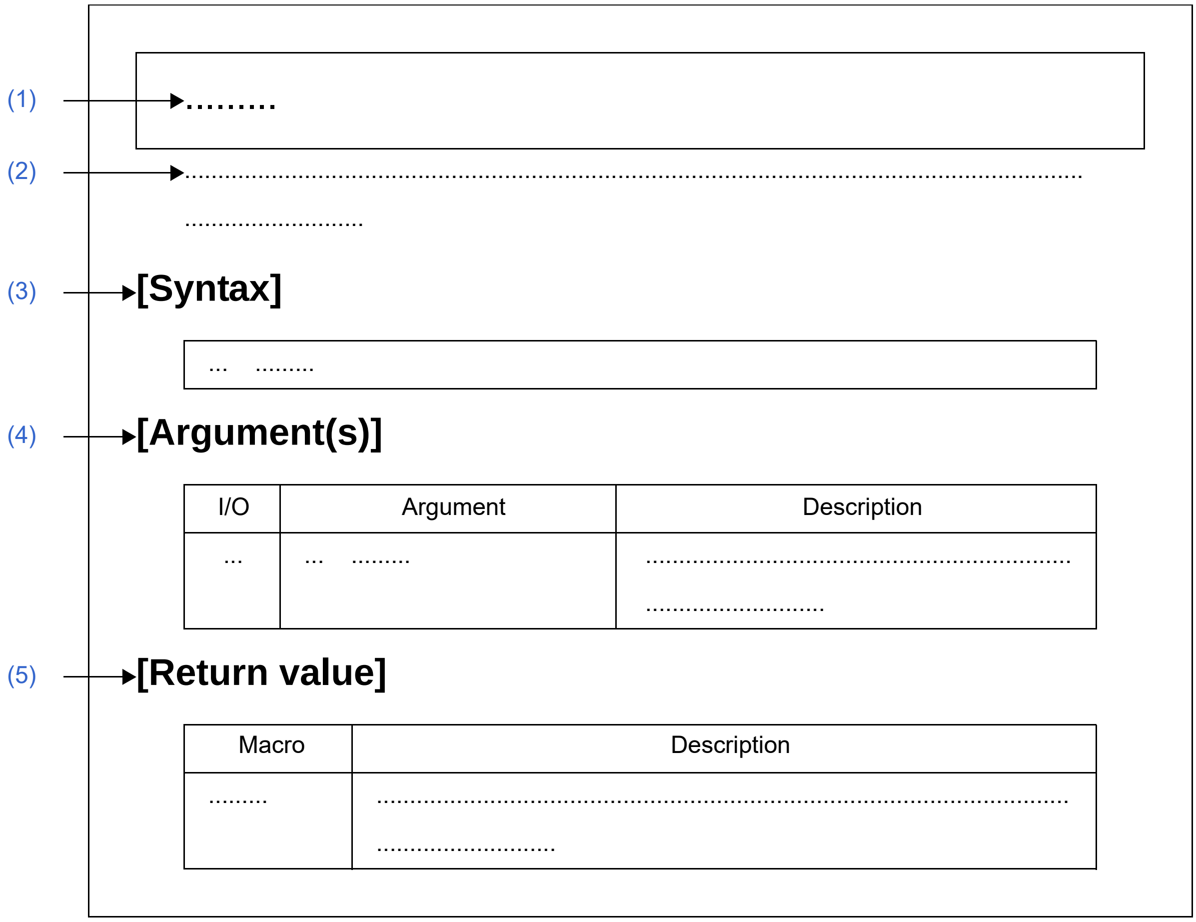
Below are the naming conventions for API functions output by the Code Generator.

- 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 3.2 API Functions: [Common]

API Function Name	Function
R_MPC_Create	Performs initialization necessary to control the multi-function pin controller.
R_MPC_Create_UserInit	Performs user-defined initialization relating to the multi-function pin controller.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
r_set_exception_handler	Sets FIQ exception handler.
r_fiq_handler	Performs FIQ exception.
NESTED_INTERRUPT_PUSH	Push the register value in response to the nested interrup.
NESTED_INTERRUPT_POP	Pop the register value in response to the nested interrup.

R_MPC_Create

Performs initialization necessary to control the multi-function pin controller.

[Syntax]

```
void    R_MPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MPC_Create_UserInit

Performs user-defined initialization relating to the multi-function pin controller.

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

[Syntax]

```
void    R_MPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark Call this API function from the startup routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_MAIN_UserInit

Performs user-defined initialization.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_Systeminit

Performs initialization necessary to control the various peripheral functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_set_exception_handler

Sets exception handler.

[Syntax]

```
void    r_set_exception_handler ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_fiq_handler

Performs FIQ handler.

[Syntax]

```
void    r_fiq_handler ( void );
```

[Argument(s)]

None.

[Return value]

None.

NESTED_INTERRUPT_PUSH

Push the register value in response to the nested interrup.

[Syntax]

```
void    NESTED_INTERRUPT_PUSH ( void );
```

[Argument(s)]

None.

[Return value]

None.

NESTED_INTERRUPT_POP

Pop the register value in response to the nested interrupt.

[Syntax]

```
void    NESTED_INTERRUPT_POP ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.2 Clock generator

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

Table 3.3 API Functions: [Clock Generator]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).
r_cgc_ostde_interrupt	Performs processing in response to the main clock oscillation stop detection interrupt.

R_CGC_Create

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_cgc_ostde_interrupt

Performs processing in response to the main clock oscillation stop detection interrupt.

[Syntax]

```
void    r_cgc_ostde_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.3 Interrupt Controller

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

Table 3.4 API Functions: [Inerrupt Controller]

API Function Name	Function
R_ICU_Create	Performs initialization required to control the interrupt controller.
R_ICU_Create_UserInit	Performs user-defined initialization relating to the interrupt controller.
R_ICU_IRQn_Start	Enables IRQ*n* interrupt.
R_ICU_IRQn_Stop	Disables IRQ*n* interrupt.
R_ICU_ETHPHYIn_Start	Enables Ether PHY interrupt (EHTPHY*n*).
R_ICU_ETHPHYIn_Stop	Disables Ether PHY interrupt (EHTPHY*n*).
r_icu_nmi_interrupt	Clears NMI mnon-maskable interrupt request.
r_icu_irqn_interrupt	Performs processing in response to the IRQ*n* interrupt.
r_icu_ethphyin_interrupt	Performs processing in response to the EHTPHYI*n*.

R_ICU_Create

Performs initialization required to control the interrupt controller.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt controller.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Start

Enables IRQ*n* interrupt.

[Syntax]

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

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Stop

Disables IRQ*n* interrupt.

[Syntax]

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

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_ETHPHYI*n*_Start

Enables Ether PHY interrupt (EHTPHY*n*).

[Syntax]

```
void    R_ICU_ETHPHYIn_Start ( void );
```

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_ETHPHYI*n*_Stop

Disables Ether PHY interrupt (EHTPHY*n*).

[Syntax]

```
void    R_ICU_ETHPHYIn_Stop ( void );
```

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

r_icu_nmi_interrupt

Clears NMI mnon-maskable interrupt request.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_icu_irq*n*_interrupt

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

[Syntax]

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

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

r_icu_ethphyi*n*_interrupt

Performs processing in response to the EHTPHYI*n*.

[Syntax]

```
void    r_icu_ethphyin_interrupt ( void );
```

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

3.2.4 Bus State Controller

Below is a list of API functions output by the Code Generator for Bus State Controller use.

Table 3.5 API Functions: [Bus State Controller]

API Function Name	Function
R_BSC_Create	Performs initialization required to control the bus state controller.
R_BSC_Create_UserInit	Performs user-defined initialization relating to the bus state controller.
r_bsc_bsccmi_interrupt	Performs processing in response to the compare match interrupt.
r_bsc_tostf_interrupt	Performs processing in response to the error of detecting a long access wait state caused by the external WAIT pin.
R_BSC_InitializeSDRAM	Performs initialization required to control the SDRAM.
R_BSC_SDRAMPowerDown_Start	After the SDRAM is accessed, the SDRAM enters the power-down mode.
R_BSC_SDRAMPowerDown_Stop	After the SDRAM is accessed, the SDRAM does not enters the power-down mode.
R_BSC_SDRAMDeepPowerDown_Start	After the SDRAM is accessed, the SDRAM enters the deep power-down mode.
R_BSC_SDRAMDeepPowerDown_Stop	The low-power SDRAM enters the self-refresh mode.

R_BSC_Create

Performs initialization required to control the bus state controller.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_BSC_Create_UserInit

Performs user-defined initialization relating to the bus state controller.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_bsc_bsccmi_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
void    r_bsc_bsccmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bsc_tostf_interrupt

Performs processing in response to the error of detecting a long access wait state caused by the external WAIT pin.

[Syntax]

```
void    r_bsc_tostf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_InitializeSDRAM

Performs initialization required to control the SDRAM.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_BSC_SDRAMPowerDown_Start

After the SDRAM is accessed, the SDRAM enters the power-down mode.

[Syntax]

```
void    R_BSC_SDRAMPowerDown_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_SDRAMPowerDown_Stop

After the SDRAM is accessed, the SDRAM does not enters the power-down mode.

[Syntax]

```
void    R_BSC_SDRAMPowerDown_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_SDRAMDeepPowerDown_Start

After the SDRAM is accessed, the SDRAM enters the deep power-down mode.

[Syntax]

```
void    R_BSC_SDRAMDeepPowerDown_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_SDRAMDeepPowerDown_Stop

The low-power SDRAM enters the self-refresh mode.

[Syntax]

```
void    R_BSC_SDRAMDeepPowerDown_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.5 DMA Controller

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

Table 3.6 API Functions: [DMA Controller]

API Function Name	Function
R_DMACn_Create	Performs initialization required to control the DMA controller.
R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
r_dmac0_dmaintn_interrupt	Performs processing in response to the external DMA request interrupt n of DMAC0.
r_dmac1_dmaint2_interrupt	Performs processing in response to the external DMA request interrupt 2 of DMAC1.
r_dmac0_dmasrq0_interrupt	Performs processing in response to the external DMA software activation request 0 interrupt of DMAC0.
r_dmac1_dmasrq1_interrupt	Performs processing in response to the external DMA software activation request 1 interrupt of DMAC1.
r_dmac0_dmaerr0_interrupt	Performs processing in response to the external DMA transfer error 0 interrupt of DMAC0.
r_dmac1_dmaerr1_interrupt	Performs processing in response to the external DMA transfer error 1 interrupt of DMAC1.
r_callback_dmac0_dmaintn_interrupt	Performs call-back processing in response to the external DMA request *n* interrupt (*n*=0~1) of DMAC0.
r_callback_dmac1_dmaint2_interrupt	Performs call-back processing in response to the external DMA request 2 interrupt of DMAC1.
R_DMACn_Set_SoftwareTrigger	Requests the DMA transfer.
R_DMACm_Cn_Start	Enables the DMA transfer of DMA Channel n.
R_DMACm_Cn_Stop	Halts the DMA transfer of DMA Channel n.
R_DMACm_Cn_Suspend	Suspends the DMA transfer of DMA Channel n.
R_DMACm_Cn_SuspendClear	Clears the temporary suspend status of the DMA transfer.

R_DMAC*n*_Create

Performs initialization required to control the DMA controller.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMACn_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Syntax]

```
void    R_DMACn_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_dmac0_dmaint*n*_interrupt

Performs processing in response to the external DMA request *n* interrupt of DMAC0.

[Syntax]

```
void    r_dmac0_dmaintn_interrupt ( void );
```

Remark *n* is the request signal number.

[Argument(s)]

None.

[Return value]

None.

r_dmac1_dmaint2_interrupt

Performs processing in response to the external DMA request 2 interrupt of DMAC1.

[Syntax]

```
void    r_dmac1_dmaint2_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmac0_dmasrq0_interrupt

Performs processing in response to the external DMA software activation request 0 interrupt of DMAC0.

[Syntax]

```
void    r_dmac0_dmasrq0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmac1_dmasrq1_interrupt

Performs processing in response to the external DMA software activation request 1 interrupt of DMAC1.

[Syntax]

```
void    r_dmac1_dmasrq1_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmac0_dmaerr0_interrupt

Performs processing in response to the external DMA transfer error 0 interrupt of DMAC0.

[Syntax]

```
void    r_dmac0_dmaerr0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmac1_dmaerr1_interrupt

Performs processing in response to the external DMA transfer error 1 interrupt of DMAC1.

[Syntax]

```
void    r_dmac1_dmaerr1_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_callback_dmac0_dmaint*n*_interrupt

Performs call-back processing in response to the external DMA request *n* interrupt (*n*=0~1) of DMAC0.

[Syntax]

```
void    r_callback_dmac0_dmaintn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_callback_dmac1_dmaint2_interrupt

Performs call-back processing in response to the external DMA request 2 interrupt of DMAC1.

[Syntax]

```
void    r_callback_dmac1_dmaint2_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Set_SoftwareTrigger

Requests the DMA transfer.

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*m*_C*n*_Start

Enables the DMA transfer of DMA Channel n.

[Syntax]

```
void    R_DMACm_Cn_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*m*_C*n*_Stop

Halts the DMA transfer of DMA Channel n.

[Syntax]

```
void    R_DMACm_Cn_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*m*_C*n*_Suspend

Suspends the DMA transfer of DMA Channel n.

[Syntax]

```
void    R_DMACm_Cn_Suspend ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_DMAC*m*_C*n*_SuspendClear

Clears the temporary suspend status of the DMA transfer.

[Syntax]

```
void    R_DMACm_Cn_SuspendClear ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

3.2.6 Event Link Controller

Below is a list of API functions output by the Code Generator for Event Link Controller use.

Table 3.7 API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization required to control the event link controller.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
r_elc_elcirqn_interrupt	Operation disable of the event link controller.
R_ELC_Start	Enables the ELC function.
R_ELC_Stop	Disables the ELC function.
R_ELC_GenerateSoftwareEvent	Disabled the software event.
R_ELC_Get_PortBuffern	Reads the value to the port buffer register（PDBF*n*）.
R_ELC_Set_PortBuffern	Writes the value to the port buffer register（PDBF*n*）.

R_ELC_Create

Performs initialization required to control the event link controller.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ELC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_elc_elcirq*n*_interrupt

Operation disable of the event link controller.

[Syntax]

```
void    r_elc_elcirqn_interrupt ( void );
```

Remark *n* is the port group number.

[Argument(s)]

None.

[Return value]

None.

R_ELC_Start

Enables the ELC function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Disables the ELC function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ELC_GenerateSoftwareEvent

Disabled the software event.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ELC_Get_PortBuffer*n*

Reads the value to the port buffer register（PDBF*n*）.

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const value;	The pointer in which the value of the buffer resister is stored.

Remark *n* is the port group number.

[Return value]

None.

R_ELC_Set_PortBuffer*n*

Writes the value to the port buffer register（PDBF*n*）.

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	uint8_t value;	The write value in the buffer register

Remark *n* is the port group number.

[Return value]

None.

3.2.7 I/O Ports

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

Table 3.8 API Functions: [I/O Ports]

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

R_PORT_Create

Performs initialization required to control the I/O port.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.2.8 Multi-Function Timer Pulse Unit

Below is a list of API functions output by the Code Generator for Multi-Function Timer Pulse Unit use.

Table 3.9 API Functions: [Multi-Function Timer Pulse Unit]

API Function Name	Function
R_MTU3_Create	Performs initialization required to control the multi-function timer pulse unit.
R_MTU3_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit.
r_mtu3_tgiam_interrupt	Performs processing in response to the channel*m* input capture/compare match A interrupt.
r_mtu3_tgibm_interrupt	Performs processing in response to the channel*m* input capture/compare match B interrupt.
r_mtu3_tgicm_interrupt	Performs processing in response to the channel*m* input capture/compare match C interrupt.
r_mtu3_tgidm_interrupt	Performs processing in response to the channel*m* input capture/compare match D interrupt.
r_mtu3_tgie0_interrupt	Performs processing in response to the channel0 input capture/compare match E interrupt.
r_mtu3_tgif0_interrupt	Performs processing in response to the channel0 input capture/compare match F interrupt.
r_mtu3_tcivm_interrupt	Performs processing in response to the channel*m* overflow interrupt.
r_mtu3_tcium_interrupt	Performs processing in response to the channel*m* underflow interrupt.
r_mtu3_tgiu5_interrupt	Performs processing in response to the channel5 input capture/compare match U interrupt.
r_mtu3_tgiv5_interrupt	Performs processing in response to the channel5 input capture/compare match V interrupt.
r_mtu3_tgiw5_interrupt	Performs processing in response to the channel5 input capture/compare match W interrupt.
r_mtu3_c4_tgia4_interrupt	Performs processing in response to the TGIA4 interrupt.
r_mtu3_c4_tgib4_interrupt	Performs processing in response to the TGIB4 interrupt.
r_mtu3_c4_tciv4_interrupt	Performs processing in response to the TCIV4 interrupt.
r_mtu3_c7_tgia7_interrupt	Performs processing in response to the TGIA7 interrupt.
r_mtu3_c7_tgib7_interrupt	Performs processing in response to the TGIB7 interrupt.
r_mtu3_c7_tciv7_interrupt	Performs processing in response to the TCIV7 interrupt.
R_MTU3_Cm_Start	Starts the count of the channel *m*.
R_MTU3_Cm_Stop	Ends the count of the channel *m*.

R_MTU3_Create

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_MTU3_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgia*m*_interrupt

Performs processing in response to the channel*m* input capture/compare match A interrupt.

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

[Syntax]

```
void    r_mtu3_tgiam_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgib*m*_interrupt

Performs processing in response to the channel*m* input capture/compare match B interrupt.

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

[Syntax]

```
void    r_mtu3_tgibm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgic*m*_interrupt

Performs processing in response to the channel*m* input capture/compare match C interrupt.

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

[Syntax]

```
void    r_mtu3_tgicm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgid*m*_interrupt

Performs processing in response to the channel*m* input capture/compare match D interrupt.

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

[Syntax]

```
void    r_mtu3_tgidm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgie0_interrupt

Performs processing in response to the channel0 input capture/compare match E interrupt.

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

[Syntax]

```
void    r_mtu3_tgie0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgif0_interrupt

Performs processing in response to the channel0 input capture/compare match F interrupt.

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

[Syntax]

```
void    r_mtu3_tgif0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciv*m*_interrupt

Performs processing in response to the channel*m* overflow interrupt.

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

[Syntax]

```
void    r_mtu3_tcivm_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciu*m*_interrupt

Performs processing in response to the channel*m* underflow interrupt.

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

[Syntax]

```
void    r_mtu3_tcium_interrupt ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgiu5_interrupt

Performs processing in response to the channel5 input capture/compare match U interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIU5 interrupt.

[Syntax]

```
void    r_mtu3_tgiu5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgiv5_interrupt

Performs processing in response to the channel5 input capture/compare match V interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIV5 interrupt.

[Syntax]

```
void    r_mtu3_tgiv5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgiw5_interrupt

Performs processing in response to the channel5 input capture/compare match W interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIW5 interrupt.

[Syntax]

```
void    r_mtu3_tgiw5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c4_tgia4_interrupt

Performs processing in response to the TGIA4 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIA4 interrupt.

[Syntax]

```
void    r_mtu3_c4_tgia4_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c4_tgib4_interrupt

Performs processing in response to the TGIB4 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIB4 interrupt.

[Syntax]

```
void    r_mtu3_c4_tgib4_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c4_tciv4_interrupt

Performs processing in response to the TCIV4 interrupt.

Remark This API function is called as the interrupt process corresponding to the TCIV4 interrupt.

[Syntax]

```
void    r_mtu3_c4_tciv4_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c7_tgia7_interrupt

Performs processing in response to the TGIA7 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIA7 interrupt.

[Syntax]

```
void    r_mtu3_c7_tgia7_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c7_tgib7_interrupt

Performs processing in response to the TGIB7 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIB7 interrupt.

[Syntax]

```
void    r_mtu3_c7_tgia7_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c7_tciv7_interrupt

Performs processing in response to the TCIV7 interrupt.

Remark This API function is called as the interrupt process corresponding to the TCIA7 interrupt.

[Syntax]

```
void    r_mtu3_c7_tcia7_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*m*_Start

Starts the count of the channel *m*.

[Syntax]

```
void    R_MTU3_Cm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*m*_Stop

Ends the count of the channel *m*.

[Syntax]

```
void    R_MTU3_Cm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.9 Port Output Enable 3

Below is a list of API functions output by the Code Generator for Port Output Enable use.

Table 3.10 API Functions: [Port Output Enable3]

API Function Name	Function
R_POE3_Create	Performs initialization required to control the port output enable3.
R_POE3_Create_UserInit	Performs user-defined initialization relating to the port output enable3.
r_poe3_oein_interrupt	Performs processing in response to the output enable interrupt.
R_POE3_Start	Places the pin in high-impedance state.
R_POE3_Stop	Disable the pin in high-impedance state.
R_POE3_Set_HiZ_MTUm	Places the MTU*m* pin in high-impedance state. Some value of m may cause MTU pins or GPT pins to be in high-impedance state.
R_POE3_Clear_HiZ_MTUm	Disables the MTU*m* pin in high-impedance state. Some value of m may cause MTU pins or GPT pins in high-impedance state to be disabled.
R_POE3_Set_HiZ_GPT3	Places the GPT3 pin in high-impedance state.
R_POE3_Clear_HiZ_GPT3	Disables the GPT3 pin in high-impedance state.

R_POE3_Create

Performs initialization required to control the port output enable3.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_POE3_Create_UserInit

Performs user-defined initialization relating to the port output enable3.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_poe3_oei*n*_interrupt

Performs processing in response to the output enable interrupt.

[Syntax]

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

Remark *n* is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Start

Places the pin in high-impedance state.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_POE3_Stop

Disable the pin in high-impedance state.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_MTU*m*

Places the MTU*m* pin in high-impedance state. Some value of m may cause MTU pins or GPT pins to be in high-impedance state.

[Syntax]

```
void    R_POE3_Set_HiZ_MTUm( void );
```

Remark *m* is the channel number. *m* can contain 0, 3_4, 6_7 as the value. In case of *m*=3_4, it specifies both MTU3 and MTU4 or GPT0~GPT2. In case of *m*=6_7, it specifies both MTU6 and MTU7.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_MTU*m*

Disables the MTU*m* pin in high-impedance state. Some value of m may cause MTU pins or GPT pins in high-impedance state to be disabled.

[Syntax]

```
void    R_POE3_Clear_HiZ_MTUm( void );
```

Remark *m* is the channel number. *m* can contain 0, 3_4, 6_7 as the value. In case of *m*=3_4, it specifies both MTU3 and MTU4 or GPT0~GPT2. In case of *m*=6_7, it specifies both MTU6 and MTU7.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_GPT3

Places the GPT3 pin in high-impedance state.

[Syntax]

```
void    R_POE3_Set_HiZ_GPT3( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_GPT3

Disables the GPT3 pin in high-impedance state.

[Syntax]

```
void    R_POE3_Clear_HiZ_GPT3( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.10 General PWM Timer

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

Table 3.11 API Functions: [General PWM Timer]

API Function Name	Function
R_GPT_Create	Performs initialization required to control the general PWM timer.
R_GPT_Create_UserInit	Performs user-defined initialization relating to the general PWM timer.
r_gpt_etgin_interrupt	Performs processing in response to the interrupt source ETGIN (External trigger falling input).
r_gpt_etgip_interrupt	Performs processing in response to the interrupt source ETGIP (External trigger rising input).
r_gpt_gtcian_interrupt	Performs processing in response to the interrupt source GTCIA*n* (GPT*n*.GTCCRA input capture/compare match).
r_gpt_gtcibn_interrupt	Performs processing in response to the interrupt source GTCIB*n* (GPT*n*.GTCCRB input capture/compare match).
r_gpt_gtcicn_interrupt	Performs processing in response to the interrupt source GTCIC*n* (GPT*n*.GTCCRC compare match).
r_gpt_gtcidn_interrupt	Performs processing in response to the interrupt source GTCID*n* (GPT*n*.GTCCRD compare match).
r_gpt_gtcien_interrupt	Performs processing in response to the interrupt source GTCIE*n* (GPT*n*.GTCCRE compare match).
r_gpt_gtcifn_interrupt	Performs processing in response to the interrupt source GTCIF*n* (GPT*n*.GTCCRF compare match).
r_gpt_gdten_interrupt	Performs processing in response to the interrupt source GDTE*n* (dead time error).
r_gpt_gtcivn_interrupt	Performs processing in response to the interrupt source GTCIV*n* (GPT*n*.GTCNT overflow or GPT*n*.GTPR compare match).
r_gpt_gtciun_interrupt	Performs processing in response to the interrupt source GTCIU*n* (GPT*n*.GTCNT underflow).
R_GPTn_Start	Starts GPT*n*.GTCNT count.
R_GPTn_Stop	Ends GPT*n*.GTCNT count.
R_GPTn_HardwareStart	Enables the GPT*n* interrupt(except the interrupt source GDTE*n* (dead time error)).
R_GPTn_HardwareStop	Disables the GPT*n* interrupt(except the interrupt source GDTE*n* (dead time error)).

R_GPT_Create

Performs initialization required to control the general PWM timer.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_GPT_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgin_interrupt

Performs processing in response to the interrupt source ETGIN (External trigger falling input).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgip_interrupt

Performs processing in response to the interrupt source ETGIP (External trigger rising input).

[Syntax]

```
void    r_gpt_etgip_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcia*n*_interrupt

Performs processing in response to the interrupt source GTCIA*n* (GPT*n*.GTCCRA input capture/compare match).

[Syntax]

```
void    r_gpt_gtcian_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcib*n*_interrupt

Performs processing in response to the interrupt source GTCIB*n* (GPT*n*.GTCCRB input capture/compare match).

[Syntax]

```
void    r_gpt_gtcibn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcic*n*_interrupt

Performs processing in response to the interrupt source GTCIC*n* (GPT*n*.GTCCRC compare match).

[Syntax]

```
void    r_gpt_gtcicn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcid*n*_interrupt

Performs processing in response to the interrupt source GTCID*n* (GPT*n*.GTCCRD compare match).

[Syntax]

```
void    r_gpt_gtcidn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcie*n*_interrupt

Performs processing in response to the interrupt source GTCIE*n* (GPT*n*.GTCCRE compare match).

[Syntax]

```
void    r_gpt_gtcien_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtcif*n*_interrupt

Performs processing in response to the interrupt source GTCIF*n* (GPT*n*.GTCCRF compare match).

[Syntax]

```
void    r_gpt_gtcifn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gdte*n*_interrupt

Performs processing in response to the interrupt source GDTE*n* (dead time error).

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciv*n*_interrupt

Performs processing in response to the interrupt source GTCIV*n* (GPT*n*.GTCNT overflow or GPT*n*.GTPR compare match).

[Syntax]

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

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciu*n*_interrupt

Performs processing in response to the interrupt source GTCIU*n* (GPT*n*.GTCNT underflow.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Start

Starts GPT*n*.GTCNT counter.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Stop

Ends GPT*n*.GTCNT counter.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStart

Enables the GPT*n* interrupt(except the interrupt source GDTE*n* (dead time error)).

Enables the GPT*n* interrupt of the following interrupt sources.

GPT interrupt sources		GPT*n* interrupt
GTCIA*n*	GPT*n*.GTCCRA input capture/compare match	Channel*n* input capture/compare match A interrupt
GTCIB*n*	GPT*n*.GTCCRB input capture/compare match	Channel*n* input capture/compare match B interrupt
GTCIC*n*	GPT*n*.GTCCRC compare match	Channel*n* compare match C interrupt
GTCID*n*	GPT*n*.GTCCRD compare match	Channel*n* compare match D interrupt
GTCIE*n*	GPT*n*.GTCCRE compare match	Channel*n* compare match E interrupt
GTCIF*n*	GPT*n*.GTCCRF compare match	Channel*n* compare match F interrupt
GTCIV*n*	GPT*n*.GTCNT overflow	Channel*n* overflow interrupt
GTCIU*n*	GPT*n*.GTCNT underflow	Channel*n* underflow interrupt

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStop

Disables the GPT*n* interrupt(except the interrupt source GDTE*n* (dead time error)).

Disables the GPT*n* interrupt of the following interrupt sources.

GPT interrupt sources		GPT*n* interrupt
GTCIA*n*	GPT*n*.GTCCRA input capture/compare match	Channel*n* input capture/compare match A interrupt
GTCIB*n*	GPT*n*.GTCCRB input capture/compare match	Channel*n* input capture/compare match B interrupt
GTCIC*n*	GPT*n*.GTCCRC compare match	Channel*n* compare match C interrupt
GTCID*n*	GPT*n*.GTCCRD compare match	Channel*n* compare match D interrupt
GTCIE*n*	GPT*n*.GTCCRE compare match	Channel*n* compare match E interrupt
GTCIF*n*	GPT*n*.GTCCRF compare match	Channel*n* compare match F interrupt
GTCIV*n*	GPT*n*.GTCNT overflow	Channel*n* overflow interrupt
GTCIU*n*	GPT*n*.GTCNT underflow	Channel*n* underflow interrupt

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.11 16-Bit Timer Pulse Unit

Below is a list of API functions output by the Code Generator for 16-Bit Timer Pulse Unit use.

Table 3.12 API Functions: [16-Bit Timer Pulse Unit]

API Function Name	Function
R_TPU_Create	Performs initialization required to control the 16-bit timer pulse unit.
R_TPU_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer pulse unit.
r_tpu_tgina_interrupt	Performs processing in response to the interrupt source TGI*n*A (TPU*n*.TGRA input capture/compare match).
r_tpu_tginb_interrupt	Performs processing in response to the interrupt source TGI*n*B (TPU*n*.TGRB input capture/compare match).
r_tpu_tginc_interrupt	Performs processing in response to the interrupt source TGI*n*C (TPU*n*.TGRC input capture/compare match).
r_tpu_tgind_interrupt	Performs processing in response to the interrupt source TGI*n*D (TPU*n*.TGRD input capture/compare match).
r_tpu_tcinv_interrupt	Performs processing in response to the interrupt source TCI*n*V (TPU*n*.TCNT overflow).
r_tpu_tcinu_interrupt	Performs processing in response to the interrupt source TCI*n*U (TPU*n*.TCNT underflow).
R_TPUn_Start	Starts TCNT count of TPU*n.*
R_TPUn_Stop	Ends TCNT count of TPU*n.*

R_TPU_Create

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_TPU_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*n*a_interrupt

Performs processing in response to the interrupt source TGI*n*A (TPU*n*.TGRA input capture/compare match).

[Syntax]

```
void    r_tpu_tgina_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*n*b_interrupt

Performs processing in response to the interrupt source TGI*n*B (TPU*n*.TGRB input capture/compare match).

[Syntax]

```
void    r_tpu_tginb_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*n*c_interrupt

Performs processing in response to the interrupt source TGI*n*C (TPU*n*.TGRC input capture/compare match).

[Syntax]

```
void    r_tpu_tgInc_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*n*d_interrupt

Performs processing in response to the interrupt source TGI*n*D (TPU*n*.TGRD input capture/compare match).

[Syntax]

```
void    r_tpu_tgind_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*v_interrupt

Performs processing in response to the interrupt source TCI*n*V (TPU*n*.TCNT overflow).

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*u_interrupt

Performs processing in response to the interrupt source TCI*n*U (TPU*n*.TCNT underflow).

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Start

Starts TCNT count of TPU*n*.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Stop

Ends TCNT count of TPU*n*.

[Syntax]

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

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.12 Programmable Pulse Generator

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

Table 3.13 API Functions: [Programmable Pulse Generator]

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

R_PPG_Create

Performs initialization required to control the programmable pulse generator.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PPG_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.2.13 Compare Match Timer

Below is a list of API functions output by the Code Generator for Compare Match Timer use.

Table 3.14 API Functions: [Compare Match Timer]

API Function Name	Function
R_CMTn_Create	Performs initialization required to control the compare match timer.
R_CMTn_Create_UserInit	Performs user-defined initialization relating to the compare match timer.
r_cmt_cmin_interrupt	Performs processing in response to the interrupt source CMI*n* (CMCNT*n* counter and CMCOR*n* register compare match).
R_CMTn_Start	Starts CMCNT*n* count.
R_CMTn_Stop	Ends CMCNT*n* count.

R_CMT*n*_Create

Performs initialization required to control the compare match timer.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Create_UserInit

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmt_cmi*n*_interrupt

Performs processing in response to the interrupt source CMI*n* (CMCNT*n* counter and CMCOR*n* register compare match).

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Start

Starts CMCNT*n* count.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Stop

Ends CMCNT*n* count.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.14 Compare Match Timer W

Below is a list of API functions output by the Code Generator for Compare Match Timer W use.

Table 3.15 API Functions: [Compare Match Timer W]

API Function Name	Function
R_CMTWm_Create	Performs initialization required to control the compare match timer W.
R_CMTWm_Create_UserInit	Performs user-defined initialization relating to the compare match timer W.
r_cmtw_cmwim_interrupt	Performs processing in response to the interrupt source CMWI (compare match).
r_cmtw_icnim_interrupt	Performs processing in response to the interrupt source IC*n*I*m* (iutput compare*n*).
r_cmtw_ocnim_interrupt	Performs processing in response to the interrupt source OC*n*I*m* (output compare*n*).
R_CMTWm_Start	Starts CMWCNT*m* count.
R_CMTWm_Stop	Ends CMWCNT*m* count.

R_CMTW*m*_Create

Performs initialization required to control the compare match timer W.

[Syntax]

```
void    R_CMTWm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*m*_Create_UserInit

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

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

[Syntax]

```
void    R_CMTWm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_cmwi*m*_interrupt

Performs processing in response to the interrupt source CMWI (compare match).

[Syntax]

```
void    r_cmtw_cmwim_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_ic*nim*_interrupt

Performs processing in response to the interrupt source IC*n*I*m* (iutput compare*n*).

[Syntax]

```
void    r_cmtw_icnim_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_oc*nim*_interrupt

Performs processing in response to the interrupt source OC*n*I*m* (output compare*n*).

[Syntax]

```
void    r_cmtw_ocnim_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_CMTW*m*_Start

Starts CMWCNT*m* count.

[Syntax]

```
void    R_CMTWm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*m*_Stop

Ends CMWCNT*m* count.

[Syntax]

```
void    R_CMTWm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.15 Watchdog Timer

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

Table 3.16 API Functions: [Watchdog Timer]

API Function Name	Function
R_WDTn_Create	Performs initialization required to control the watchdog timer.
R_WDTn_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
r_wdtn_undff_refef_interrupt	Performs processing in response to the WDT overflow / refresh error.
R_WDTn_Restart	Clears the watchdog timer counter and resumes counting.

R_WDT*n*_Create

Performs initialization required to control the watchdog timer.

[Syntax]

```
void    R_WDTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_WDT*n*_Create_UserInit

Performs user-defined initialization relating to the watchdog timer.

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

[Syntax]

```
void    R_WDTn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_wdt*n*_undff_refef_interrupt

Performs processing in response to the WDT overflow / refresh error.

[Syntax]

```
void    r_wdtn_undff_refef_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_WDT*n*_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```
void    R_WDTn_Restart ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.16 Independent Watchdog Timer

Below is a list of API functions output by the Code Generator for Independent Watchdog Timer use.

Table 3.17 API Functions: [Independent Watchdog Timer]

API Function Name	Function
R_IWDT_Create	Performs initialization required to control the independent watchdog timer.
R_IWDT_Create_UserInit	Performs user-defined initialization relating to the independent watchdog timer.
r_iwdt_undff_refef_interrupt	Performs processing in response to the IWDT overflow / refresh error.
R_IWDT_Restart	Clears the independent watchdog timer counter and resumes counting.

R_IWDT_Create

Performs initialization required to control the independent watchdog timer.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_iwdt_undff_refef_interrupt

Performs processing in response to the IWDT overflow / refresh error.

[Syntax]

```
void    r_iwdt_undff_refef_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Restart

Clears the independent watchdog timer counter and resumes counting.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.2.17 Serial Communications Interface with FIFO

Below is a list of API functions output by the Code Generator for Serial Communication Interface with FIFO use.

Table 3.18 API Functions: [Serial Communication Interface with FIFO]

API Function Name	Function
R_SCIFAn_Create	Performs initialization necessary to control the serial communications interface with FIFO.
R_SCIFAn_Create_UserInit	Performs user-defined initialization relating to the serial communications interface with FIFO.
r_scifan_txifn_interrupt	Performs processing in response to the interrupt source TXI (FIFO data empty interrupts).
r_scifan_rxifn_interrupt	Performs processing in response to the interrupt source RXI (receive FIFO data full interrupts).
r_scifan_brifn_interrupt	Performs processing in response to the interrupt source BRI (break or overrun interrupts) and ERI (framing error or parity error interrupts).
r_scifan_drifn_interrupt	Performs processing in response to the interrupt source DRI (receive data ready interrupts) and TEI (transmit-end interrupts).
r_scifan_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scifan_callback_receiveend	Performs processing in response to the receive FIFO data full interrupts.
r_scifan_callback_error	Performs processing in response to the error interrupts.
R_SCIFAn_Start	Starts serial communications interface with FIFO.
R_SCIFAn_Stop	Ends serial communications interface with FIFO.
R_SCIFAn_Serial_Send	Starts serial communications interface with FIFO transmission (asynchronous mode).
R_SCIFAn_Serial_Receive	Starts serial communications interface with FIFO reception (asynchronous mode).
R_SCIFAn_Serial_Send_Receive	Starts serial communications interface with FIFO transmission/ reception (clock synchronous mode).

R_SCIFA*n*_Create

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Create_UserInit

Performs user-defined initialization relating to the serial communications interface with FIFO.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_txif*n*_interrupt

Performs processing in response to the interrupt source TXI (FIFO data empty interrupts).

[Syntax]

```
void    r_scifan_txifn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_rxif*n*_interrupt

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

[Syntax]

```
void    r_scifan_rxifn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_brif*n*_interrupt

Performs processing in response to the interrupt source BRI (break or overrun interrupts) and ERI (framing error or parity error interrupts).

[Syntax]

```
void    r_scifan_brifn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_drif*n*_interrupt

Performs processing in response to the interrupt source DRI (receive data ready interrupts) and TEI (transmit-end interrupts).

[Syntax]

```
void    r_scifan_drifn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_receiveend

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_error

Performs processing in response to the error interrupts.

Remark This API function is called as the r_scifan_erifn_interrupt or r_scifan_brifn_interrupt callback routine.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Start

Starts serial communications interface with FIFO.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Stop

Ends serial communications interface with FIFO.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Serial_Send

Starts FIFO embedded SCI transmission (asynchronous mode).

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

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

[Syntax]

```
MD_STATUS   R_SCIFAn_Serial_Send ( const uint8_t * txbuf, uint16_t txnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Receive

Starts FIFO embedded SCI reception (asynchronous mode).

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

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

[Syntax]

```
MD_STATUS   R_SCIFAn_Serial_Receive ( uint8_t * rxbuf, uint16_t rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Send_Receive

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

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

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

3.2.18 I^{2}C Bus Interface

Below is a list of API functions output by the Code Generator for I^{2}C Bus Interface use.

Table 3.19 API Functions: [I^{2}C Bus Interface]

API Function Name	Function
R_RIICn_Create	Performs initialization necessary to control the I^{2}C bus interface.
R_RIICn_Create_UserInit	Performs user-defined initialization relating to the I^{2}C bus interface.
r_riicn_error_interrupt	Performs processing in response to the transfer error/event generation interrupts (EEI).
r_riicn_receive_interrupt	Performs processing in response to the receive data full interrupts (RXI).
r_riicn_transmit_interrupt	Performs processing in response to the transmit data empty interrupts (TXI).
r_riicn_transmitend_interrupt	Performs processing in response to the transmit end interrupts (TEI).
R_RIICn_Start	Starts RIIC communication.
R_RIICn_Stop	Ends RIIC communication.
R_RIICn_Master_Send	Starts RIIC master transmission. Generates Stop condition when transmission is completed.
R_RIICn_Master_Send_Without_Stop	Starts RIIC master transmission. Not generates Stop condition when transmission is completed.
R_RIICn_Master_Receive	Starts RIIC master reception.
R_RIICn_Slave_Send	Starts RIIC slave transmission.
R_RIICn_Slave_Receive	Starts RIIC slave reception.
R_RIICn_StartCondition	Issues the start condition and causes a transfer error and an event generation interrupt (EEI).
R_RIICn_StopCondition	Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).
r_riicn_callback_receiveerror	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.
r_riicn_callback_transmitend	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.
r_riicn_callback_receiveend	Of the interrupt processing for transfer error/event generation interrupts (EEI), processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive is performed.

R_RIIC*n*_Create

Performs initialization necessary to control the I²C bus interface.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Create_UserInit

Performs user-defined initialization relating to the I^2C bus interface.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_error_interrupt

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_receive_interrupt

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmit_interrupt

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmitend_interrupt

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Start

Starts RIIC communication.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Stop

Ends RIIC communication.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Master_Send

Starts RIIC master transmission.

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

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t adr;`	Slave address
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_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIICn_Master_Send_Without_Stop

Starts RIIC master transmission.
Not generates Stop condition when transmission is completed.

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

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t adr;`	Slave address
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_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIICn_Master_Receive

Starts RIIC master reception.

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

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

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

[Syntax]

```
MD_STATUS R_RIICn_Master_Receive(uint16_t adr, uint8_t * rx_buf, uint16_t rx_num);
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

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

R_RIIC*n*_Slave_Send

Starts RIIC slave transmission.

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

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

[Syntax]

```
void    R_RIICn_Slave_Send ( const uint8_t * tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	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]

None.

R_RIIC*n*_Slave_Receive

Starts RIIC slave reception.

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

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

[Syntax]

```
void    R_RIICn_Slave_Receive ( uint8_t * rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

R_RIIC*n*_StartCondition

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

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_StopCondition

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveerror

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

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

[Syntax]

```
static  void    r_riicn_callback_receiveerror ( MD_STATUS status );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

r_riicn_callback_transmitend

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveend

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.19 Serial Peripheral Interface

Below is a list of API functions output by the Code Generator for Serial Peripheral Interface use.

Table 3.20 API Functions: [Serial Peripheral Interface]

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

R_RSPI*n*_Create

Performs initialization necessary to control the serial peripheral interface.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Create_UserInit

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

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_receive_interrupt

Performs processing in response to the receive buffer full interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_transmit_interrupt

Performs processing in response to the transmit buffer empty interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_error_interrupt

Performs processing in response to the RSPI error interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_idle_interrupt

Performs processing in response to the RSPI idle interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Start

Starts RSPI communication.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Stop

Ends RSPI communication.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Send

Starts RSPI transmission.

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

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

[Syntax]

```
MD_STATUS   R_RSPIn_Send (const uint32_t * tx_buf, uint16_t tx_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`const uint32_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 *tx_num* specification

R_RSPI*n*_Send_Receive

Starts RSPI transmission/reception.

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

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

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

[Syntax]

```
MD_STATUS   R_RSPIn_Send_Receive (const uint32_t * tx_buf, uint16_t tx_num, uint32_t *
rx_buf);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`const uint32_t * tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send/receive
O	`uint32_t * rx_buf;`	Pointer to a buffer to store the reception data

[Return value]

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

r_rspi*n*_callback_receiveend

Performs processing in response to the receive buffer full interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_callback_error

Performs processing in response to the RSPI error interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

r_rspi*n*_callback_transmitend

Performs processing in response to the RSPI idle interrupts.

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

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.2.20 SPI Multi I/O Bus Controller

Below is a list of API functions output by the Code Generator for SPI Multi I/O Bus Controller use.

Table 3.21 API Functions: [SPI Multi I/O Bus Controller]

API Function Name	Function
R_SPIBSC_Create	Performs initialization required to control the SPI multi I/O bus controller.
R_SPIBSC_Create_UserInit	Performs user-defined initialization relating to the SPI multi I/O bus controller.
R_SPIBSC_EAVUpperAddressChange	Changes the address of higher 8-bit in the address 32-bit output.
R_SPIBSC_SPIRead	Receives data.
R_SPIBSC_SPIWrite	Transmits data.
R_SPIBSC_SPIRead_Write	Receives and transmits data.

R_SPIBSC_Create

Performs initialization required to control the SPI multi I/O bus controller.

[Syntax]

```
void    R_SPIBSC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPIBSC_Create_UserInit

Performs user-defined initialization relating to the SPI multi I/O bus controller.

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

[Syntax]

```
void    R_SPIBSC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPIBSC_EAVUpperAddressChange

Changes the address of higher 8-bit in the address 32-bit output.

[Syntax]

```
void R_SPIBSC_EAVUpperAddressChange(uint8_t const up_adr);
```

[Argument(s)]

I/O	Argument	Description
I	`uint8_t const up_adr;`	Address of higher 8-bit in the address 32-bit output

[Return value]

None.

R_SPIBSC_SPIRead

Receives the data with the SPI-mode.

[Syntax]

```
MD_STATUS R_SPIBSC_SPIRead(st_spibsc_spiparam const param, uint32_t rx_data);
```

[Argument(s)]

I/O	Argument	Description
I	`st_spibsc_spiparam const param;`	Parameter of the SPI-mode (Read / Write)
O	`uint32_t rx_data;`	Received data

[Return value]

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

R_SPIBSC_SPIWrite

Transmits the data with the SPI-mode.

[Syntax]

```
MD_STATUS R_SPIBSC_SPIWrite(st_spibsc_spiparam const param, uint32_t const tx_data);
```

[Argument(s)]

I/O	Argument	Description
I	`st_spibsc_spiparam const param;`	Parameter of the SPI-mode (Read / Write)
I	`uint32_t const tx_data;`	Transmitted data

[Return value]

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

R_SPIBSC_SPIRead_Write

Receives and transmits the data with the SPI-mode.

[Syntax]

```
MD_STATUS R_SPIBSC_SPIRead_Write(st_spibsc_spiparam const param, uint32_t rx_data,
uint32_t const tx_data);
```

[Argument(s)]

I/O	Argument	Description
I	`st_spibsc_spiparam const param;`	Parameter of the SPI-mode (Read / Write)
O	`uint32_t const rx_data;`	Received data
I	`uint32_t const tx_data;`	Transmitted data

[Return value]

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

3.2.21 CRC Operation Units

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

Table 3.22 API Functions: [CRC Operation Units]

API Function Name	Function
R_CRC_SetCRC8_2F	Initializes the CRC operation unit for 8-bit CRC calculation (CCRC8-0x2F).
R_CRC_SetCRC8_SAE	Initializes the CRC operation unit for 8-bit CRC calculation (CRC8-SAE-J1850).
R_CRC_SetCRC16_CCITT	Initializes the CRC operation unit for 16-bit CRC calculation (CRC16-CCITT).
R_CRC_SetCRC32_ETHER	Initializes the CRC operation unit for 32-bit CRC calculation (CRC32_ETHERNET).
R_CRC_Input_Data	Sets the initial value of the data from which the CRC is to be calculated.
R_CRC_Get_Result	Gets the result of operation.

R_CRC_SetCRC8_2F

Initializes the CRC operation unit for 8-bit CRC calculation (CCRC8-0x2F).

[Syntax]

```
void    R_CRC_SetCRC8_2F ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_SetCRC8_SAE

Initializes the CRC operation unit for 8-bit CRC calculation (CRC8-SAE-J1850).

[Syntax]

```
void    R_CRC_SetCRC8_SAE ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_SetCRC16_CCITT

Initializes the CRC operation unit for 16-bit CRC calculation (CRC16-CCITT).

[Syntax]

```
void    R_CRC_SetCRC16_CCITT ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_SetCRC32_ETHER

Initializes the CRC operation unit for 32-bit CRC calculation (CRC32_ETHERNET).

[Syntax]

```
void    R_CRC_SetCRC32_ETHER ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_Input_Data

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

[Syntax]

```
void    R_CRC_Input_Data ( uint32_t data );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t data;`	The initial value of the data from which the CRC is to be calculated

[Return value]

None.

R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```
void    R_CRC_Get_Result ( uint32_t * const result );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const result;`	Pointer to the location where the result of operation is stored

[Return value]

None.

3.2.22 ΔΣ Interface

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

Table 3.23 API Functions: [ΔΣ Interface]

API Function Name	Function
R_DSMIF_Create	Performs initialization required to control the ΔΣ interface.
R_DSMIF_Create_UserInit	Performs user-defined initialization relating to the ΔΣ interface.
R_DSMIF_UVW_Start	Filtering is started of ΔΣ interface (unit0 (U, V, W)).
R_DSMIF_UVW_Stop	Filtering is stopped of ΔΣ interface (unit0 (U, V, W)).
R_DSMIF_X_Start	Filtering is started of ΔΣ interface (unit1 (X)).
R_DSMIF_X_Stop	Filtering is stopped of ΔΣ interface (unit1 (X)).
r_dsmif_uvw_overcurrent_interrupt	Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).
r_dsmif_uvw_totalcurrent_interrupt	Performs processing in response to the totalcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).
r_dsmif_uvw_shortcircuit_interrupt	Performs processing in response to the short circuit interrupt of ΔΣ interface (unit0 (U, V, W)).
r_dsmif_x_overcurrent_interrupt	Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit1 (X)).
r_dsmif_x_shortcircuit_interrupt	Performs processing in response to the short circuit interrupt of ΔΣ interface (unit1 (X)).

R_DSMIF_Create

Performs initialization required to control the ΔΣ interface.

[Syntax]

```
void    R_DSMIF_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSMIF_Create_UserInit

Performs user-defined initialization relating to the ΔΣ interface.

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

[Syntax]

```
void    R_DSMIF_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSMIF_UVW_Start

Filtering is started of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```
void    R_DSMIF_UVW_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSMIF_UVW_Stop

Filtering is stopped of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```
void    R_DSMIF_UVW_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSMIF_X_Start

Filtering is started of ΔΣ interface (unit1 (X)).

[Syntax]

```
void    R_DSMIF_X_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSMIF_X_Stop

Filtering is stopped of ΔΣ interface (unit1 (X)).

[Syntax]

```
void    R_DSMIF_X_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsmif_uvw_overcurrent_interrupt

Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```
void    r_dsmif_uvw_overcurrent_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsmif_uvw_totalcurrent_interrupt

Performs processing in response to the totalcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```
void    r_dsmif_uvw_totalcurrent_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsmif_uvw_shortcircuit_interrupt

Performs processing in response to the shortcircuit interrupt of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```
void    r_dsmif_uvw_shortcircuit_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsmif_x_overcurrent_interrupt

Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit1 (X)).

[Syntax]

```
void    r_dsmif_x_overcurrent_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsmif_x_shortcircuit_interrupt

Performs processing in response to the short circuit interrupt of ΔΣ interface (unit1 (X)).

[Syntax]

```
void    r_dsmif_x_shortcircuit_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.23 Memory Protection Unit

Below is a list of API functions output by the Code Generator for Memory Protection Unit use

表 3.24 API Functions: [Memory Protection Unit]

API Function Name	Function
R_MPU_Create	Performs initialization required to control the Memory Protection Unit.

R_MPU_Create

Performs initialization required to control the Memory Protection Unit.

[Syntax]

```
void    R_MPU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.24 Error Control Module

Below is a list of API functions output by the Code Generator for Error Control Module use.

Table 3.25 API Functions: [Error Control Module]

API Function Name	Function
R_ECM_Create	Performs initialization required to control the error control module.
R_ECM_Create_UserInit	Performs user-defined initialization relating to the error control module.
r_ecm_nmi_interrupt	Performs processing in response to the ECM non-maskable interrupt.
r_ecm_compareerror_interrupt	Performs processing in response to the ECM compare error interrupt.
R_ECM_Pseudo_WDT0_Error_Start	Generates corresponding pseudo error of WDT overflow / refresh error (error source 1) (Cortex-R4F).
R_ECM_Pseudo_WDT0_Error_Stop	Pseudo error not generated WDT overflow / refresh error (error source 1) (Cortex-R4F).
R_ECM_Pseudo_IWDTa_Error_Start	Generates corresponding pseudo error of IWDTa overflow / refresh error (error source 3).
R_ECM_Pseudo_IWDTa_Error_Stop	Pseudo error not generated IWDTa overflow / refresh error (error source 3).
R_ECM_Pseudo_CGC_Error_Start	Enables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.
R_ECM_Pseudo_CGC_Error_Stop	Disables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.
R_ECM_Pseudo_ADC_Unit0_Error_Start	Generates corresponding pseudo error of 12-bit A/D converter unit0 overwrite interrupt (error source 24).
R_ECM_Pseudo_ADC_Unit0_Error_Stop	Pseudo error not generated 12-bit A/D converter unit0 overwrite interrupt (error source 24).
R_ECM_Pseudo_ADC_Unit1_Error_Start	Generates corresponding pseudo error of 12-bit A/D converter unit1 overwrite interrupt (error source 25).
R_ECM_Pseudo_ADC_Unit1_Error_Stop	Pseudo error not generated 12-bit A/D converter unit1 overwrite interrupt (error source 25).
R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start	Generates corresponding pseudo error of ΔΣ interface UVW overcurrent abnormality detection error (error source 26).
R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop	Pseudo error not generated ΔΣ interface UVW overcurrent abnormality detection error (error source 26).
R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Start	Generates corresponding pseudo error of ΔΣ interface UVW total current abnormality detection error (error source 27).
R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop	Pseudo error not generated ΔΣ interface UVW total current abnormality detection error (error source 27).
R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start	Generates corresponding pseudo error of ΔΣ interface UVW short circuit abnormality detection error (error source 28).
R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop	Pseudo error not generated ΔΣ interface UVW short circuit abnormality detection error (error source 28).
R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start	Generates corresponding pseudo error of ΔΣ interface X overcurrent abnormality detection error (error source 29).

API Function Name	Function
R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop	Pseudo error not generated ΔΣ interface X overcurrent abnormality detection error (error source 29).
R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start	Generates corresponding pseudo error of ΔΣ interface X short circuit abnormality detection error (error source 31).
R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop	Pseudo error not generated ΔΣ interface X short circuit abnormality detection error (error source 31).
R_ECM_Pseudo_DOC_Error_Start	Generates corresponding pseudo error of DOC operation error (error source 32).
R_ECM_Pseudo_DOC_Error_Stop	Pseudo error not generated DOC operation error (error source 32).
R_ECM_Pseudo_BSC_Error_Start	Generates corresponding pseudo error of bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).
R_ECM_Pseudo_BSC_Error_Stop	Pseudo error not generated bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).
R_ECM_Pseudo_Error35_Error_Start	Generates corresponding pseudo error of extended pseudo error 35 (error source 35).
R_ECM_Pseudo_Error35_Error_Stop	Pseudo error not generated extended pseudo error 35 (error source 35).
R_ECM_Pseudo_Error36_Error_Start	Generates corresponding pseudo error of extended pseudo error 36 (error source 36).
R_ECM_Pseudo_Error36_Error_Stop	Pseudo error not generated extended pseudo error 36 (error source 36).
R_ECM_Pseudo_Error37_Error_Start	Generates corresponding pseudo error of extended pseudo error 37 (error source 37).
R_ECM_Pseudo_Error37_Error_Stop	Pseudo error not generated extended pseudo error 37 (error source 37).
R_ECM_Pseudo_Error38_Error_Start	Generates corresponding pseudo error of extended pseudo error 38 (error source 38).
R_ECM_Pseudo_Error38_Error_Stop	Pseudo error not generated extended pseudo error 38 (error source 38).
R_ECM_Pseudo_Error39_Error_Start	Generates corresponding pseudo error of extended pseudo error 39 (error source 39).
R_ECM_Pseudo_Error39_Error_Stop	Pseudo error not generated extended pseudo error 39 (error source 39).
R_ECM_Pseudo_Error40_Error_Start	Generates corresponding pseudo error of extended pseudo error 40 (error source 40).
R_ECM_Pseudo_Error40_Error_Stop	Pseudo error not generated extended pseudo error 40 (error source 40).
R_ECM_Pseudo_Error41_Error_Start	Generates corresponding pseudo error of extended pseudo error 41 (error source 41).
R_ECM_Pseudo_Error41_Error_Stop	Pseudo error not generated extended pseudo error 41 (error source 41).
R_ECM_Pseudo_ECM_CompareError_Error_Start	Generates corresponding pseudo error of ECM compare error (error source 93).

API Function Name	Function
R_ECM_Pseudo_ECM_CompareError_Error_Stop	Pseudo error not generated ECM compare error (error source 93).
R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start	Generates corresponding pseudo error of ECM delay timer overflow (error source 94).
R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop	Pseudo error not generated ECM delay timer overflow (error source 94).

R_ECM_Create

Performs initialization required to control the error control module.

[Syntax]

```
void    R_ECM_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Create_UserInit

Performs user-defined initialization relating to the error control module.

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

[Syntax]

```
void    R_ECM_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_ecm_nmi_interrupt

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

[Syntax]

```
void    r_ecm_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_ecm_errd_interrupt

Performs processing in response to the error detection.

[Syntax]

```
void    r_ecm_errd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_ecm_compareerror_interrupt

Performs processing in response to the ECM compare error interrupt.

[Syntax]

```
void    r_ecm_compareerror_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_WDT0_Error_Start

Generates corresponding pseudo error of WDT overflow / refresh error (error source 1) (Cortex-R4F).

[Syntax]

```
void    R_ECM_Pseudo_WDT0_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_WDT0_Error_Stop

Pseudo error not generated WDT overflow / refresh error (error source 1) (Cortex-R4F).

[Syntax]

```
void    R_ECM_Pseudo_WDT0_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_IWDTa_Error_Start

Generates corresponding pseudo error of IWDTa overflow / refresh error (error source 3).

[Syntax]

```
void    R_ECM_Pseudo_IWDTa_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_IWDTa_Error_Stop

Pseudo error not generated IWDTa overflow / refresh error (error source 3).

[Syntax]

```
void    R_ECM_Pseudo_IWDTa_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_CGC_Error_Start

Enables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.

[Syntax]

```
void    R_ECM_Pseudo_CGC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_CGC_Error_Stop

Disables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.

[Syntax]

```
void    R_ECM_Pseudo_CGC_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ADC_Unit0_Error_Start

Generates corresponding pseudo error of 12-bit A/D converter unit0 overwrite interrupt (error source 24).

[Syntax]

```
void    R_ECM_Pseudo_ADC_Unit0_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ADC_Unit0_Error_Stop

Pseudo error not generated 12-bit A/D converter unit0 overwrite interrupt (error source 24).

[Syntax]

```
void    R_ECM_Pseudo_ADC_Unit0_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ADC_Unit1_Error_Start

Generates corresponding pseudo error of 12-bit A/D converter unit1 overwrite interrupt (error source 25).

[Syntax]

```
void    R_ECM_Pseudo_ADC_Unit1_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ADC_Unit1_Error_Stop

Pseudo error not generated 12-bit A/D converter unit1 overwrite interrupt (error source 25).

[Syntax]

```
void    R_ECM_Pseudo_ADC_Unit1_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start

Generates corresponding pseudo error of ΔΣ interface UVW overcurrent abnormality detection error (error source 26).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop

Pseudo error not generated ΔΣ interface UVW overcurrent abnormality detection error (error source 26).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Start

Generates corresponding pseudo error of ΔΣ interface UVW total current abnormality detection error (error source 27).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop

Pseudo error not generated ΔΣ interface UVW total current abnormality detection error (error source 27).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start

Generates corresponding pseudo error of ΔΣ interface UVW short circuit abnormality detection error (error source 28).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop

Pseudo error not generated ΔΣ interface UVW short circuit abnormality detection error (error source 28).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start

Generates corresponding pseudo error of ΔΣ interface X overcurrent abnormality detection error (error source 29).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop

Pseudo error not generated ΔΣ interface X overcurrent abnormality detection error (error source 29).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start

Generates corresponding pseudo error of ΔΣ interface X short circuit abnormality detection error (error source 31).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop

Pseudo error not generated ΔΣ interface X short circuit abnormality detection error (error source 31).

[Syntax]

```
void    R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DOC_Error_Start

Generates corresponding pseudo error of DOC operation error (error source 32).

[Syntax]

```
void    R_ECM_Pseudo_DOC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_DOC_Error_Stop

Pseudo error not generated DOC operation error (error source 32).

[Syntax]

```
void    R_ECM_Pseudo_DOC_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_BSC_Error_Start

Generates corresponding pseudo error of bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).

[Syntax]

```
void    R_ECM_Pseudo_BSC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_BSC_Error_Stop

Pseudo error not generated bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).

[Syntax]

```
void    R_ECM_Pseudo_BSC_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error35_Error_Start

Generates corresponding pseudo error of extended pseudo error 35 (error source 35).

[Syntax]

```
void    R_ECM_Pseudo_Error35_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error35_Error_Stop

Pseudo error not generated extended pseudo error 35 (error source 35).

[Syntax]

```
void    R_ECM_Pseudo_Error35_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error36_Error_Start

Generates corresponding pseudo error of extended pseudo error 36 (error source 36).

[Syntax]

```
void    R_ECM_Pseudo_Error36_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error36_Error_Stop

Pseudo error not generated extended pseudo error 36 (error source 36).

[Syntax]

```
void    R_ECM_Pseudo_Error36_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error37_Error_Start

Generates corresponding pseudo error of extended pseudo error 37 (error source 37).

[Syntax]

```
void    R_ECM_Pseudo_Error37_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error37_Error_Stop

Pseudo error not generated extended pseudo error 37 (error source 37).

[Syntax]

```
void    R_ECM_Pseudo_Error37_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error38_Error_Start

Generates corresponding pseudo error of extended pseudo error 38 (error source 38).

[Syntax]

```
void    R_ECM_Pseudo_Error38_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error38_Error_Stop

Pseudo error not generated extended pseudo error 38 (error source 38).

[Syntax]

```
void    R_ECM_Pseudo_Error38_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error39_Error_Start

Generates corresponding pseudo error of extended pseudo error 39 (error source 39).

[Syntax]

```
void    R_ECM_Pseudo_Error39_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error39_Error_Stop

Pseudo error not generated extended pseudo error 39 (error source 39).

[Syntax]

```
void    R_ECM_Pseudo_Error39_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error40_Error_Start

Generates corresponding pseudo error of extended pseudo error 40 (error source 40).

[Syntax]

```
void    R_ECM_Pseudo_Error40_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error40_Error_Stop

Pseudo error not generated extended pseudo error 40 (error source 40).

[Syntax]

```
void    R_ECM_Pseudo_Error40_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error41_Error_Start

Generates corresponding pseudo error of extended pseudo error 41 (error source 41).

[Syntax]

```
void    R_ECM_Pseudo_Error41_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_Error41_Error_Stop

Pseudo error not generated extended pseudo error 41 (error source 41).

[Syntax]

```
void    R_ECM_Pseudo_Error41_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ECM_CompareError_Error_Start

Generates corresponding pseudo error of ECM compare error (error source 93).

[Syntax]

```
void    R_ECM_Pseudo_ECM_CompareError_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ECM_CompareError_Error_Stop

Pseudo error not generated ECM compare error (error source 93).

[Syntax]

```
void    R_ECM_Pseudo_ECM_CompareError_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start

Generates corresponding pseudo error of ECM delay timer overflow (error source 94).

[Syntax]

```
void    R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop

Pseudo error not generated ECM delay timer overflow (error source 94).

[Syntax]

```
void    R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.25 12-Bit A/D Converter

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

Table 3.26 API Functions: [12-Bit A/D Converter]

API Function Name	Function
R_S12ADn_Create	Performs initialization required to control the 12-Bit A/D converter.
R_S12ADn_Create_UserInit	Performs user-defined initialization relating to the 12-Bit A/D converter.
r_s12ad_s12adn_interrupt	Performs processing in response to the A/D conversion end interrupt.
r_s12ad_s12gbadin_interrupt	Performs processing in response to the group B A/D conversion end interrupt.
r_s12ad_s12aden_interrupt	Performs processing in response to the A/D error interrupt.
R_S12ADn_Start	Starts the A/D conversion.
R_S12ADn_Stop	Ends the A/D conversion.
R_S12ADn_Get_ValueResult	Gets the result of conversion.
R_S12ADn_Set_CompareValue	Sets compare level.
r_s12ad_s12cmpn_interrupt	Performs processing in response to the compare interrupt.

R_S12AD*n*_Create

Performs initialization required to control the 12-Bit A/D converter.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Create_UserInit

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

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

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad_s12ad*n*_interrupt

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

[Syntax]

```
void    r_s12ad_s12adn_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad_s12gbadi*n*_interrupt

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

[Syntax]

```
static  void    r_s12ad_s12gbadin_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad_s12aden_interrupt

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

[Syntax]

```
void    r_s12ad_s12aden_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Start

Starts the A/D conversion.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Stop

Ends the A/D conversion.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Get_ValueResult

Gets the conversion result.

[Syntax]

```
void    R_S12ADn_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

Remark *n* is the unit number.

[Argument(s)]

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

[Return value]

None.

R_S12AD*n*_Set_CompareValue

Sets compare level.

[Syntax]

```
void    R_S12ADn_Set_CompareValue ( uint16_t  reg_value0,uint16_t  reg_value1 );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	*uint16_t reg_value0*	Register value set to the compare revel register 0
I	*uint16_t reg_value1*	Register value set to the compare revel register 1

[Return value]

None.

r_s12ad_s12cmp*n*_interrupt

Performs processing in response to the compare interrupt.

[Syntax]

```
static void    r_s12ad_s12cmpn_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

3.2.26 Data Operation Circuit

Below is a list of API functions output by the Code Generator for Data Operation Circuit use.

Table 3.27 API Functions: [Data Operation Circuit]

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

R_DOC_Create

Performs initialization necessary to control the data operation circuit.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DOC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_doc_dopcf_interrupt

Performs processing in response to the data operation circuit interrupt.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

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

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

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

[Syntax]

```
void    R_DOC_SetMode ( doc_mode_t mode, uint16_t value );
```

[Argument(s)]

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

[Return value]

None.

R_DOC_WriteData

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

[Syntax]

```
void    R_DOC_WriteData ( uint16_t data );
```

[Argument(s)]

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

[Return value]

None.

R_DOC_GetResult

Gets the result of operation.

[Syntax]

```
void    R_DOC_GetResult ( uint16_t * const data );
```

[Argument(s)]

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

[Return value]

None.

R_DOC_ClearFlag

Clears the data operation circuit flag.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Apr 01, 2015	-	First Edition issued
1.01	Dec 01, 2017	-	Correction of errors, Creation of the description for omissions
1.02	Mar 13, 2020	23 - 24	Added following functions. NESTED_INTERRUPT_PUSH NESTED_INTERRUPT_POP
		172 178 - 179	Deleted following functions. r_scifan_teifn_interrupt r_scifan_erifn_interrupt
		188, 198	Added following functions. R_RIICn_Master_Send_Without_Stop
		238, 245 - 249	Added following functions. r_dsmif_uvw_overcurrent_interrupt r_dsmif_uvw_totalcurrent_interrupt r_dsmif_uvw_shortcircuit_interrupt r_dsmif_x_overcurrent_interrupt r_dsmif_x_shortcircuit_interrupt
		251 - 252	Added following functions. R_MPU_Create

AP4 User's Manual:
RZ API Reference

Publication Date: Rev.1.00 Apr 1, 2015
Rev.1.02 Mar 13, 2020

Published by: Renesas Electronics Corporation

AP4

Renesas Electronics Corporation