AP4
Integrated Development Environment
User's Manual: RH850 API Reference

Target Device
RH850 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).
1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

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

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

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

All trademarks or registered trademarks in this document are the property of their respective owners.
# TABLE OF CONTENTS

1. **GENERAL** .................................................................................................................. 5  
   1.1 Overview .................................................................................................................. 5  
   1.2 Features ................................................................................................................... 5  
   1.3 About RH850 Code Generator tool .......................................................................... 5  
2. **OUTPUT FILE** .......................................................................................................... 6  
   2.1 Description .............................................................................................................. 6  
3. **API FUNCTIONS** ...................................................................................................... 13  
   3.1 Overview ................................................................................................................. 13  
   3.2 Function Reference ................................................................................................. 13  
      3.2.1 Common .......................................................................................................... 15  
      3.2.2 Clock controller ............................................................................................. 19  
      3.2.3 Port functions ............................................................................................... 24  
      3.2.4 Interrupt ........................................................................................................ 27  
      3.2.5 DMAC ............................................................................................................. 41  
      3.2.6 DTS ................................................................................................................. 55  
      3.2.7 Clock Serial Interface G ............................................................................... 71  
      3.2.8 Clock Serial Interface H ............................................................................... 84  
      3.2.9 Serial Communication Interface 3 ................................................................. 104  
      3.2.10 UART Interface ............................................................................................ 120  
      3.2.11 Window Watchdog Timer ............................................................................ 133  
      3.2.12 OS Timer ....................................................................................................... 138  
      3.2.13 Advanced Timer Unit IV ................................................................................ 145  
      3.2.14 Timer Array Unit B ....................................................................................... 186  
      3.2.15 Timer Array Unit D ....................................................................................... 193  
      3.2.16 Timer Array Unit J ....................................................................................... 200  
      3.2.17 Timer Option ................................................................................................. 207  
      3.2.18 Peripheral Interconnection ........................................................................... 214  
      3.2.19 A/D converter .............................................................................................. 218  
      3.2.20 Delta-Sigma AD converter ............................................................................ 240  
      3.2.21 Digital Filter ................................................................................................... 248  
      3.2.22 Data CRC ...................................................................................................... 262  
      3.2.23 Real-Time Clock .......................................................................................... 270  
      3.2.24 Key Return .................................................................................................... 292  
      3.2.25 Stand-By Controller ...................................................................................... 298  

Revision Record .................................................................................................................. 310
1. GENERAL

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

1.1 Overview

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

1.2 Features

Code Generator tool has the following features.

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

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

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

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

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

1.3 About RH850 Code Generator tool

The RH850 Code Generator tool has the following original spec.

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

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

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

This appendix describes the files output by the Code Generator.

2.1 Description

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

Table 2.1 Output File List

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>r_cg_main.c</td>
<td>main R_MAIN_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_systeminit.c</td>
<td>R_SystemInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_macrodriver.h</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_intvector.c</td>
<td>-</td>
</tr>
<tr>
<td>Clock controller</td>
<td>r_cg_cgc.c</td>
<td>R_CGC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CGC_CK_Output_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CGC_CK_Output_Disable</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc_user.c</td>
<td>R_CGC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc.h</td>
<td>-</td>
</tr>
<tr>
<td>Port functions</td>
<td>r_cg_port.c</td>
<td>R_PORT_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_port_user.c</td>
<td>R_PORT_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_port.h</td>
<td>-</td>
</tr>
<tr>
<td>Interrupt</td>
<td>r_cg_intc.c</td>
<td>R_INTC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_IRQn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_IRQn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SINTn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SINTn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SINTn_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_INTPn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_INTPn_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_intc_user.c</td>
<td>R_INTC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_nmi_interrupt</td>
<td>r_irqn_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_sintn_interrupt</td>
<td>r_intpn_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_intpninterrupt</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_intc.h</td>
<td>-</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMAC</td>
<td>r_cg_dmac.c</td>
<td>R_DMACn_Create&lt;br&gt;R_DMACn_Suspend&lt;br&gt;R_DMACn_Resume&lt;br&gt;R_DMACnm_Create&lt;br&gt;R_DMACnm_Start&lt;br&gt;R_DMACnm_Stop&lt;br&gt;R_DMACnm_Set_SoftwareTrigger&lt;br&gt;R_DMACnm_Suspend&lt;br&gt;R_DMACnm_Resume</td>
</tr>
<tr>
<td>DMAC</td>
<td>r_cg_dmac_user.c</td>
<td>R_DMAC_Create_UserInit&lt;br&gt;r_dmacnm_interrupt&lt;br&gt;r_dmacnm_callback_transfer_completion&lt;br&gt;r_dmacnm_callback_transfer_count_match</td>
</tr>
<tr>
<td>DMAC</td>
<td>r_cg_dmac.h</td>
<td>--</td>
</tr>
<tr>
<td>DTS</td>
<td>r_cg_dts.c</td>
<td>R_DTS_Create&lt;br&gt;R_DTS_Suspend&lt;br&gt;R_DTS_Resume&lt;br&gt;R_DTS_All_Stop&lt;br&gt;R_DTSx_y_Stop_Interrupt&lt;br&gt;R_DTSm_Create&lt;br&gt;R_DTSm_Start&lt;br&gt;R_DTSm_Stop&lt;br&gt;R_DTSm_Set_SoftwareTrigger&lt;br&gt;R_DTSm_Suspend&lt;br&gt;R_DTSm_Resume</td>
</tr>
<tr>
<td>DTS</td>
<td>r_cg_dts_user.c</td>
<td>R_DTS_Create_UserInit&lt;br&gt;R_DTSm_Create_UserInit&lt;br&gt;r_dtsx_y_transfer_match_interrupt&lt;br&gt;r_dtsx_y_transfer_completion_interrupt</td>
</tr>
<tr>
<td>DTS</td>
<td>r_cg_dts.h</td>
<td>--</td>
</tr>
<tr>
<td>Clock Serial Interface G</td>
<td>r_cg_csig.c</td>
<td>R_CSIgm_Create&lt;br&gt;R_CSIgm_Start&lt;br&gt;R_CSIgm_Stop&lt;br&gt;R_CSIgm_Send&lt;br&gt;R_CSIgm_Receive</td>
</tr>
<tr>
<td>Clock Serial Interface G</td>
<td>r_cg_csig_user.c</td>
<td>R_CSIgm_Create_UserInit&lt;br&gt;r_csigm_interrupt_receive&lt;br&gt;r_csigm_interrupt_error&lt;br&gt;r_csigm_interrupt_send&lt;br&gt;r_csigm_callback_receiveend&lt;br&gt;r_csigm_callback_sendend&lt;br&gt;r_csigm_callback_error</td>
</tr>
<tr>
<td>Clock Serial Interface G</td>
<td>r_cg_csig.h</td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Clock Serial Interface H | r_cg_csih.c | R_CSIHm_Create  
R_CSIHm_Start  
R_CSIHm_Stop  
R_CSIHm_Master_Send  
R_CSIHm_Master_Receive  
R_CSIHm_Slave_Send  
R_CSIHm_Slave_Receive  
R_CSIHm_Extended_Data_Master_Send  
R_CSIHm_Extended_Data_Master_Receive  
R_CSIHm_Extended_Data_Slave_Send  
R_CSIHm_Extended_Data_Slave_Receive  |
|                     | r_cg_csih_user.c | R_CSIHm_Create_UserInit  
r_csihm_interrupt_receive  
r_csihm_interrupt_error  
r_csihm_interrupt_send  
r_csihm_interrupt_jobend  
r_csihm_callback_receiveend  
r_csihm_callback_sendend  
r_csihm_callback_error  |
|                     | r_cg_csih.h      | --                                                                                 |
| Serial Communication Interface 3 | r_cg_sci3.c | R_SCI3m_Create  
R_SCI3m_Start  
R_SCI3m_Stop  
R_SCI3m_Send  
R_SCI3m_Receive  
R_SCI3m_Multiprocessor_Send  
R_SCI3m_Multiprocessor_Receive  |
|                     | r_cg_sci3_user.c | R_SCI3m_Create_UserInit  
r_sci3m_interrupt_receive  
r_sci3m_interrupt_error  
r_sci3m_interrupt_send  
r_sci3m_interrupt_jobend  
r_sci3m_callback_receiveend  
r_sci3m_callback_sendend  
r_sci3m_callback_error  |
|                     | r_cg_sci3.h      | --                                                                                 |
| UART Interface      | r_cg_uart.c      | R_UARTm_Create  
R_UARTm_Start  
R_UARTm_Stop  
R_UARTm_Send  
R_UARTm_Receive  |
|                     | r_cg_uart_user.c | R_UARTm_Create_UserInit  
r_uartm_interrupt_receive  
r_uartm_interrupt_error  
r_uartm_interrupt_jobend  
r_uartm_callback_receiveend  
r_uartm_callback_sendend  
r_uartm_callback_error  |
<p>|                     | r_cg_uart.h      | --                                                                                 |</p>
<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window Watchdog Timer</td>
<td>r_cg_wdt.c</td>
<td>R_WDTm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_WDTm_Restart</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt_user.c</td>
<td>R_WDTm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_wdtm_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt.h</td>
<td>—</td>
</tr>
<tr>
<td>OS Timer</td>
<td>r_cg_ostm.c</td>
<td>R_OSTMm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_OSTMm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_OSTMm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_OSTMm_Set_CompareValue</td>
</tr>
<tr>
<td></td>
<td>r_cg_ostm_user.c</td>
<td>R_OSTMm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ostmm_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_ostm.h</td>
<td>—</td>
</tr>
<tr>
<td>Advanced Timer Unit IV</td>
<td>r_cg_atuiv.c</td>
<td>R_ATUIV_Common_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerkn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknk_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknk_OperationOff</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Get_CaptureValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Set_Compare_Match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Set_One_Shot_Pulse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Forced_Compare_Match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Forced_Output_Compare_Match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknk_Start_Down_Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknk_Get_InputCapturex</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknx_xpin_Output_Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknx_xpin_Output_Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknx_xpin_Output_High</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Get_Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Get_PWM_Measure_Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Get_Measure_Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerknm_Reset_FIFO</td>
</tr>
<tr>
<td></td>
<td>r_cg_atuiv_user.c</td>
<td>R_ATUIV_Common_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATUIV_Timerkn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_overflow_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_underflow_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_icnx_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_ocrnx_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_tcntnx_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_cmfxn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_callback_ocrc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_callback_grc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_callback_duty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_fifo_overflow_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_fifo_datafull_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_atuiv_timerknm_tcntk_overflow_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_atuiv.h</td>
<td>—</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Timer Array Unit B</td>
<td>r_cg_taub.c</td>
<td>R_TAUBn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUBn_Channelm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUBn_Channelm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUBn_Channelm_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>r_cg_taub_user.c</td>
<td>R_TAUBn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_taubn_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_taub.h</td>
<td>--</td>
</tr>
<tr>
<td>Timer Array Unit D</td>
<td>r_cg_taud.c</td>
<td>R_TAUDn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUDn_Channelm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUDn_Channelm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUDn_Channelm_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>r_cg_taud_user.c</td>
<td>R_TAUDn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_taudn_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_taud.h</td>
<td>--</td>
</tr>
<tr>
<td>Timer Array Unit J</td>
<td>r_cg_tauj.c</td>
<td>R_TAUJn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUJn_Channelm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUJn_Channelm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAUJn_Channelm_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>r_cg_tauj_user.c</td>
<td>R_TAUJn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_taujn_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_tauj.h</td>
<td>--</td>
</tr>
<tr>
<td>Timer Option</td>
<td>r_cg_tapa.c</td>
<td>R_TAPAm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAPAm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAPAm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAPAm_Trigger_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TAPAm_Trigger_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_tapa_user.c</td>
<td>R_TAPAm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_tapa.h</td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Interconnection</td>
<td>r_cg_pic.c</td>
<td>R_PICn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_PICn_Timer.SyncStart</td>
</tr>
<tr>
<td></td>
<td>r_cg_pic_user.c</td>
<td>R_PICn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_pic.h</td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>A/D converter</strong></td>
<td>r_cg_adc.c</td>
<td>R_ADCn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_Halt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_SetMultiplexerCommand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_GetFloatingPointDataResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_TimerStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_TimerStop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ADCSummation_Groupm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ADCSummation_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ADCSummation_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADC.SyncStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADC.SyncTimerStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_ScanGroupm_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ADCn_TH_Groupx_Start</td>
</tr>
<tr>
<td>r_cg_adc_user.c</td>
<td></td>
<td>R_ADCn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_adc_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_adc_scan_groupm_end_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_adc_multiplexer_request_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_adc_adc_summation_channelm_end_interrupt</td>
</tr>
<tr>
<td>r_cg_adc.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Delta-Sigma AD converter</strong></td>
<td>r_cg_dsadc.c</td>
<td>R_DSADC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DSADC_SyncStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DSADCm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DSADCm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DSADCm_GetResult</td>
</tr>
<tr>
<td>r_cg_dsadc_user.c</td>
<td></td>
<td>R_DSADC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dsadc_error_interrupt</td>
</tr>
<tr>
<td>r_cg_dsadc.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Digital Filter</strong></td>
<td>r_cg_dfe.c</td>
<td>R_DFE_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Set_SoftwareData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Generate_SoftwareTrigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Channelm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Channelm_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Channelm_Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Channelm_GetResult</td>
</tr>
<tr>
<td>r_cg_dfe_user.c</td>
<td></td>
<td>R_DFE_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dfe_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DFE_Channelm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dfe_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dfe_channelm_callback_output_data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dfe_channelm_callback_condition_match</td>
</tr>
<tr>
<td>r_cg_dfe.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data CRC</strong></td>
<td>r_cg_dcra.c</td>
<td>R_DCRAn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DCRAn_Input32bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DCRAn_Input16bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DCRAn_Input8bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DCRAn_GetResult_32bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DCRAn_GetResult_16bitData</td>
</tr>
<tr>
<td>r_cg_dcra_user.c</td>
<td></td>
<td>R_DCRAn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r_cg_dcra.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Real-Time Clock</td>
<td>r_cg_rtca.c</td>
<td>R_RTC_Create&lt;br&gt;R_RTC_Start&lt;br&gt;R_RTC_Stop&lt;br&gt;R_RTC_Set_HourSystem&lt;br&gt;R_RTC_Set_CounterValue&lt;br&gt;R_RTC_Get_CounterValue&lt;br&gt;R_RTC_Set_AlarmOn&lt;br&gt;R_RTC_Set_AlarmOff&lt;br&gt;R_RTC_Set_AlarmValue&lt;br&gt;R_RTC_Get_AlarmValue&lt;br&gt;R_RTC_Set(ConstPeriod)InterruptOn&lt;br&gt;R_RTC_Set(ConstPeriod)InterruptOff&lt;br&gt;R_RTC_Set(1second)InterruptOn&lt;br&gt;R_RTC_Set(1second)InterruptOff&lt;br&gt;R_RTC_Set_RTC1HZOn&lt;br&gt;R_RTC_Set_RTC1HZOff</td>
</tr>
<tr>
<td></td>
<td>r_cg_rtca_user.c</td>
<td>R_RTC_Create_UserInit&lt;br&gt;r_rtc_interrupt_periodic&lt;br&gt;r_rtc_interrupt_alarm&lt;br&gt;r_rtc_interrupt_1second</td>
</tr>
<tr>
<td></td>
<td>r_cg_rtca.h</td>
<td>—</td>
</tr>
<tr>
<td>Key Return</td>
<td>r_cg_key.c</td>
<td>R_KEY_Create&lt;br&gt;R_KEY_Start&lt;br&gt;R_KEY_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_key_user.c</td>
<td>R_KEY_Create_UserInit&lt;br&gt;r_key_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_key.h</td>
<td>—</td>
</tr>
<tr>
<td>Stand-By Controller</td>
<td>r_cg_stbc.c</td>
<td>R_STBC_Start_Stop_Mode&lt;br&gt;R_STBC_Prepare_Stop_Mode&lt;br&gt;R_STBC_Start_Deep_Stop_Mode&lt;br&gt;R_STBC_Prepare_Deep_Stop_Mode&lt;br&gt;R_STBC_Deep_Stop_Loop</td>
</tr>
<tr>
<td></td>
<td>r_cg_stbc_user.c</td>
<td>R_STBC_Prepare_Stop_Mode_Set_Peripheral&lt;br&gt;R_STBC_Prepare_Stop_Mode_Set_Interrupt&lt;br&gt;R_STBC_Prepare_Stop_Mode_Set_Clock_Mask&lt;br&gt;R_STBC_Prepare_Stop_Mode_Set_Clock_Source&lt;br&gt;R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral&lt;br&gt;R_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_stbc.h</td>
<td>—</td>
</tr>
</tbody>
</table>
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)]
   I/O  | Argument  | Description
   --- | --------- | -------------------------------
   ... | ...       | ........................................
   ... | ...       | ........................................

5. [Return value]
   Macro  | Description
   ------ | -------------------------------
   ........ | ........................................
   ........ | ........................................
(4) [Argument(s)]
API function arguments are explained in the following format.

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

(a) Macro
Macro of return value

(b) Description
Description of return value
3.2.1 Common

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

Table 3.1 API Functions: [Common]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>This is a main function.</td>
</tr>
<tr>
<td>R_MAIN_UserInit</td>
<td>Performs user-defined initialization.</td>
</tr>
<tr>
<td>R_SystemInit</td>
<td>Performs initialization necessary to control the various peripheral functions.</td>
</tr>
</tbody>
</table>
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

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

Argument(s)

None.

Return value

None.
R_SystemInit

Performs initialization necessary to control the various peripheral functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
3.2.2 Clock controller

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CGC_Create</td>
<td>Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).</td>
</tr>
<tr>
<td>R_CGC_Create_UserInit</td>
<td>Performs user-define initialization relating to the clock generator (include reset function, on-chip debug function, etc.).</td>
</tr>
<tr>
<td>R_CGC_CK_Output_Enable</td>
<td>Enables output of the CK pin.</td>
</tr>
<tr>
<td>R_CGC_CK_Output_Enable</td>
<td>Enables output of the CK pin.</td>
</tr>
</tbody>
</table>
### R_CGC_Create

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_CGC_Create_UserInit**

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

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

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
R_CGC_CK_Output_Enable

Enables output of the CK pin.

[Syntax]
void R_CGC_CK_Output_Enable(void);

[Argument(s)]
None.

[Return value]
None.
**R_CGC_CK_Output_Disable**

Disables output of the CK pin.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.3 Port functions

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_PORT_Create</td>
<td>Performs initialization necessary to control the port functions.</td>
</tr>
<tr>
<td>R_PORT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the port functions.</td>
</tr>
</tbody>
</table>
R_PORT_Create

Performs initialization necessary to control the port functions.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_PORT_Create_UserInit**

Performs user-defined initialization relating to the port functions.

**Remark** This API functions is called as the `R_PORT_Create` callback routine.

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
3.2.4 Interrupt

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

Table 3.4 API Functions: [Interrupt]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_INTC_Create</td>
<td>Performs initialization necessary to control the interrupt functions.</td>
</tr>
<tr>
<td>R_INTC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the interrupt functions.</td>
</tr>
<tr>
<td>R_IRQn_Start</td>
<td>Enables the IRQn interrupts.</td>
</tr>
<tr>
<td>R_IRQn_Stop</td>
<td>Disables the IRQn interrupts.</td>
</tr>
<tr>
<td>R_SINTn_Start</td>
<td>Enables the software interrupts (SINTn).</td>
</tr>
<tr>
<td>R_SINTn_Stop</td>
<td>Disables the software interrupts (SINTn).</td>
</tr>
<tr>
<td>R_SINTn_TriggerOn</td>
<td>Software interrupt registers increments the value of a counter.</td>
</tr>
<tr>
<td>R_INTPn_Start</td>
<td>Enables the INTPn interrupts.</td>
</tr>
<tr>
<td>R_INTPn_Stop</td>
<td>Disables the INTPn interrupts.</td>
</tr>
<tr>
<td>r_nmi_interrupt</td>
<td>Performs processing in response to the NMI interrupt.</td>
</tr>
<tr>
<td>r_irqn_interrupt</td>
<td>Performs processing in response to the IRQn interrupt.</td>
</tr>
<tr>
<td>r_sintn_interrupt</td>
<td>Performs processing in response to the SINTn interrupt.</td>
</tr>
<tr>
<td>r_intpn_interrupt</td>
<td>Performs processing in response to the INTPn interrupt.</td>
</tr>
</tbody>
</table>
**R_INTC_Create**

Performs initialization necessary to control the interrupt functions.

**[Syntax]**

```c
void R_INTC_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_INTC_Create_UserInit**

Performs user-defined initialization relating to the interrupt functions.

**Remark** This API function is called as the R_INTC_Create callback routine.

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
Enables the IRQ\textit{n} interrupts.

**Syntax**

```c
void    R_IRQ\textit{n}_Start ( void );
```

Remark  \textit{n} is the interrupt factor number.

**Argument(s)**

None.

**Return value**

None.
R_IRQn_Stop

Disables the IRQn interrupts.

[Syntax]

void R_IRQn_Stop ( void );

Remark  \( n \) is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.
Enables the software interrupt (SINTn).

**[Syntax]**

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

Remark  \( n \) is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables the software interrupt (SINTn).

**Syntax**

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

Remark  
\( n \) is the interrupt factor number.

**Argument(s)**

None.

**Return value**

None.
**R_SINTn_TriggerOn**

Software interrupt registers increments the value of a counter.

**[Syntax]**

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

**Remark**

$n$ is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_nmi_interrupt

Performs processing in response to the NMI interrupt.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
Performs processing in response to the IRQn interrupt.

**[Syntax]**

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

**Remark**

$n$ is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
[Syntax]

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

Remark  

\( n \) is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.
### R_INTPn_Start

Enables the INTPn interrupts.

#### [Syntax]

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

**Remark**

$n$ is the interrupt factor number.

#### [Argument(s)]

None.

#### [Return value]

None.
Disables the INTPn interrupts.

**Syntax**

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

**Remark**

$n$ is the interrupt factor number.

**Argument(s)**

None.

**Return value**

None.
r_intp\_n\_interrupt

Performs processing in response to the INTPn interrupt.

**[Syntax]**

```c
void r_intp\_n\_interrupt ( void );
```

**Remark**

\( n \) is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.5 DMAC

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DM4N_Create</td>
<td>Performs initialization necessary to control the DMACn functions.</td>
</tr>
<tr>
<td>R_DM4N_Suspend</td>
<td>Suspend DMAC transfer for all channels.</td>
</tr>
<tr>
<td>R_DM4N_Resume</td>
<td>Resume DMAC transfer for all channels.</td>
</tr>
<tr>
<td>R_DM4_Create_UserInit</td>
<td>Performs user-defined initialization relating to the DMACn functions.</td>
</tr>
<tr>
<td>r_dmacnm_interrupt</td>
<td>Performs processing in response to the DMACn channelm interrupt.</td>
</tr>
<tr>
<td>r_dmacnm_callback_transfer_completion</td>
<td>Performs processing in response to the DMAC transfer end interrupt.</td>
</tr>
<tr>
<td>r_dmacnm_callback_transfer_count_match</td>
<td>Performs processing in response to the DMAC transfer count match interrupt.</td>
</tr>
<tr>
<td>R_DM4nm_Create</td>
<td>Performs initialization necessary to control the DMACn channelm functions.</td>
</tr>
<tr>
<td>R_DM4nm_Start</td>
<td>Enables the DMACn channelm transfer.</td>
</tr>
<tr>
<td>R_DM4nm_Stop</td>
<td>Disables the DMACn channelm transfer.</td>
</tr>
<tr>
<td>R_DM4nm_Suspend</td>
<td>Suspend DMACn channelm transfer.</td>
</tr>
<tr>
<td>R_DM4nm_Resume</td>
<td>Resume DMACn channelm transfer.</td>
</tr>
</tbody>
</table>
**R_DMACn_Create**

Performs initialization necessary to control the DNAC functions.

**[Syntax]**

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

Remark  

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACn_Suspend

Suspend DMAC transfer for all channels.

[Syntax]

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

[Remark]
n is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_DMACn_Resume

Resume DMAC transfer for all channels.

[Syntax]

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

Remark $n$ is the unit number.

[Argument(s)]
None.

[Return value]
None.
**R_DMAC_Create_UserInit**

Performs user-defined initialization relating to the DMAC functions.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_dmacnm_interrupt

Performs processing in response to the DMAC\textsubscript{n} channel\textsubscript{m} interrupt.

[Syntax]

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

Remark \( n \) is unit number, \( m \) is channel number.

[Argument(s)]

None.

[Return value]

None.
**r_dmacnm_callback_transfer_completion**

Performs processing in response to the DMA transfer end interrupt.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_dmacntm_callback_transfer_count_match

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

[Syntax]

```c
void r_dmacntm_callback_transfer_count_match(void);
```

Remark

\( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs initialization necessary to control the DMAC_n channel_m functions.

**Syntax**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACnm_Start

Enables the DMACn channelm transfer.

[Syntax]

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

Remark  

- \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Disables the DMAC\textsubscript{n} channel\textsubscript{m} transfer.

**Syntax**

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

*Remark*  
\textit{n} is the unit number, \textit{m} is the channel number.

**Argument(s)**

None.

**Return value**

None.
Generates the DMAC\textsubscript{n} channel\textsubscript{m} transfer request.

**[Syntax]**

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

**Remark** \textit{n} is the unit number, \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Suspend DMACn channel m transfer.

**[Syntax]**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Resume DMAC\textsuperscript{n} channel\textsuperscript{m} transfer.

**[Syntax]**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.6 DTS

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

Table 3.6 API Functions: [DTS]

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

Performs initialization necessary to control the DTS functions.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DTS_Create_UserInit**

Performs user-defined initialization relating to the DTS functions.

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

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
R_DTS_Suspend

Suspend DTS transfer.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
### R_DTS_Resume

Resume DTS transfer.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_DTS_All_Stop

Stop all DTS transfer.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_DTSx_y_Stop_Interrupt

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

[Syntax]

```c
void R_DTSx_y_Stop_Interrupt ( void );
```

Remark  

x and y is channel number.

[Argument(s)]

None.

[Return value]

None.
**R_DTSm_Create**

Performs initialization necessary to control the DTS channel \( m \) functions.

**[Syntax]**

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

**Remark**

\( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Enables the DTS channel\textit{m} transfer.

**[Syntax]**

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

Remark \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables the DTS channel \( m \) transfer.

**[Syntax]**

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

**Remark**  \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Generates the DTS channel \( m \) transfer request.

**[Syntax]**

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

**Remark**

\( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_DTSm_Suspend

Suspend DTS channel \( m \) transfer.

#### Syntax

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

#### Remark

\( m \) is the channel number.

#### Argument(s)

None.

#### Return value

None.
Resume DTS channel $m$ transfer.

### [Syntax]

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

Remark $m$ is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
Perform user-defined initialization relating to the DTS channel $m$ functions.

**Remark** This API function is called as the **R_DTSm_Create** callback routine.

**Syntax**

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

**Remark** $m$ is the channel number.

**Argument(s)**

None.

**Return value**

None.
**r_dtsx_y_transfer_match_interrupt**

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

**[Syntax]**

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

Remark  
- x and y is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_dtsx_y_transfer_completion_interrupt**

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

**[Syntax]**

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

**Remark**

x and y is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.7 Clock Serial Interface G

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

Table 3.7  API Functions: [Clock serial interface G]

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

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

**[Syntax]**

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

Remark  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_CSIGm_Create_UserInit**

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

**Remark**  
This API function is called as the `R_CSIGm_Create` callback routine.

**Syntax**

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

**Remark**  
`m` is the channel number.

**Argument(s)**

None.

**Return value**

None.
r_csigm_interrupt_receive

Performs processing in response to the CSIG reception interrupt.

[Syntax]

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

Remark  \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csigm_interrupt_error

Performs processing in response to the CSIG error interrupt.

[Syntax]

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

Remark  
\( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the CSIG communication interrupt.

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csigm_callback_receiveend

Performs processing in response to the CSIG reception interrupt.

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

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_csigm_callback_sendend**

Performs processing in response to the CSIG communication interrupt.

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

**[Syntax]**

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

**Remark**  
`m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_csig\_m\_callback\_error**

Performs processing in response to the CSIG error interrupt.

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

### Syntax

```c
#include "r_cg_macrodriver.h"
void r_csig\_m\_callback\_error ( uint8\_t err\_type );
```

**Remark**
\(m\) is the channel number.

### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t err_type;</td>
<td>Trigger for CSIG error interrupt&lt;br&gt;0000x0x1B : Overrun error&lt;br&gt;0000x01xB : Parity error&lt;br&gt;000010xxB : Data consistency check error</td>
</tr>
</tbody>
</table>

### Return value

None.
Sets CSIG communication to standby mode.

**Syntax**

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

Remark  

\( m \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
### R_CSIGm_Stop

Ends CSIG communication.

#### [Syntax]

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

Remark  
$m$ is the channel number.

#### [Argument(s)]

None.

#### [Return value]

None.
**R_CSIgm_Send**

Start CSIG data transmission.

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

**Remark 2.** `R_CSIgm_Start` must be called before this API function is called.

**[Syntax]**

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

**Remark**  
`m` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>const uint16_t * tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_num;</code></td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Start CSIG data reception.

Remark 1. This API function performs 2 byte-level CSIG reception the number of times specified by the argument \textit{rx\_num}, and stores the data in the buffer specified by the argument \textit{rx\_buf}.

Remark 2. Starts after this API function is called, and \texttt{R\_CSIGm\_Start} is then called.

[Syntax]

```c
#include "r_cg_macrodriver.h"
MD_STATUS R\_CSIGm\_Receive (\ const\ uint16\_t *\ rx\_buf,\ uint16\_t\ rx\_num );
```

Remark \textit{m} is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>\textit{const\ uint16_t *\ rx_buf};</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>\textit{uint16_t\ rx_num};</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGBERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

\textit{Remark \textit{m} is the channel number.}
### 3.2.8 Clock Serial Interface H

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

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

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

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_CSIHm_Create_UserInit**

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

Remark  This API functions is called as the `R_CSIHm_Create` callback routine.

### [Syntax]

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

Remark  \( m \) is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
r_csihm_interrupt_receive

Performs processing in response to the CSIH reception interrupt.

[Syntax]

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

Remark

$m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csihm_interrupt_error

Performs processing in response to the CSIХ error interrupt.

[Syntax]

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

Remark  
$m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csihm_interrupt_send

Performs processing in response to the CSIH communication interrupt.

[Syntax]

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

Remark  
$m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csihm_interrupt_jobend

Performs processing in response to the CSIH job end interrupt.

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### r_csihm_callback_receiveend

**Performs processing in response to the CSIH reception interrupt.**

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

**[Syntax]**

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

**Remark** `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_csihm_callback_sendend

Performs processing in response to the CSIH communication interrupt.

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

[Syntax]

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

Remark: $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_csihm_callback_error

Performs processing in response to the CSIH error interrupt.

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

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t err_type;</td>
<td>Trigger for CSIH error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x0xB : Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x01B : Parity error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000010xB : Data consistency check error</td>
</tr>
</tbody>
</table>

[Return value]

None.
Sets CSIH communication to standby mode.

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Ends CSIH communication.

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Start CSI data transmission by master mode.

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

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

### Syntax

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

Remark: `m` is the channel number.

### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>const uint16_t * tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_num;</code></td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>I</td>
<td><code>uint32_t chipId;</code></td>
<td>Set chip select</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Start CSIH data reception by master mode.

Remark 1. This API function performs 2 byte-level CSIH reception the number of times specified by the argument \textit{rx\_num}, and stores the data in the buffer specified by the argument \textit{rx\_buf}.

Remark 2. Starts after this API function is called, and \textit{R\_CSIHm\_Start} is then called.

**[Syntax]**

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

Remark $m$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>\textit{const uint16_t * rx_buf};</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>\textit{uint16_t rx_num};</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td>I</td>
<td>\textit{uint32_t chipId};</td>
<td>Set chip select</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
### R_CSIHm_Slave_Send

Start CSIH data transmission by slave mode.

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

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

**Syntax**

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

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>const uint16_t * tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_num;</code></td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Start CSIH data reception by slave mode.

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

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

**[Syntax]**

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

Remark  \( m \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td><code>const uint16_t * rx_buf;</code></td>
<td>Pointer to a buffer storing the received data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t rx_num;</code></td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
## R_CSIHm_Extended_Data_Master_Send

Start CSIH extended data transmission by master mode.

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

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

### Syntax

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_CSIHm_Extended_Data_Master_Send (const uint16_t * tx_buf, uint16_t tx_bit_num, uint32_t chipId);
```

**Remark** 
*m* is the channel number.

### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint16_t * tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_bit_num;</td>
<td>Number of bits to send</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t chipId;</td>
<td>Set chip select</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
R_CSIHm_Extended_Data_Master_Receive

Start CSIH extended data reception by master mode.

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

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

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>const int16_t * rx_buf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>int16_t rx_bit_num;</td>
<td>Number of bits to receive</td>
</tr>
<tr>
<td>I</td>
<td>int32_t chipId;</td>
<td>Set chip select</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
### Start CSIH extended data transmission by slave mode.

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

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

#### Syntax

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

**Remark** $m$ is the channel number.

#### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>const uint16_t * tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_bit_num;</code></td>
<td>Number of bits to send</td>
</tr>
</tbody>
</table>

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
R_CSIHm_Extended_Data_Slave_Receive

Start CSIH extended data reception by slave mode.

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

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

**[Syntax]**

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_CSIHm_Slave_Receive ( const uint16_t * rx_buf, uint16_t rx_bit_num );
```

Remark  $m$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>const uint16_t * rx_buf;</td>
<td>Pointer to a buffer storing the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_bit_num;</td>
<td>Number of bits to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
3.2.9 Serial Communication Interface 3

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

Table 3.9 API Functions: [Serial communication interface 3]

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

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

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_SCI3m_Create_UserInit

Performs user-defined initialization relating to the serial communication interface 3 functions.
Remark This API functions is called as the R_SCI3m_Create callback routine.

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_sci3m_interrupt_receive

Performs processing in response to the SCI reception interrupt.

[Syntax]

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

Remark  

\( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_sci3m_interrupt_error**

Performs processing in response to the SCI error interrupt.

**[Syntax]**

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

**Remark**  
\( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_sci3m_interrupt_send

Performs processing in response to the SCI transmit interrupt.

**[Syntax]**

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

**Remark**

\( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
AP4 3. API FUNCTIONS

r_sci3m_interrupt_sendend

Performs processing in response to the SCI transmit end interrupt.

[Syntax]

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

Remark  \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_sci3m_callback_receiveend**

Performs processing in response to the SCI reception interrupt.

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

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**  
None.

**[Return value]**  
None.
r_sci3m_callback_sendend

Performs processing in response to the SCI transmit interrupt.

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

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### r_sci3m_callback_error

Performs processing in response to the SCI error interrupt.

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

**[Syntax]**

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

**Remark**
`m` is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
Sets SCI3 communication to standby mode.

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_SCI3m_Stop

Ends SCI3 communication.

[Syntax]

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

Remark  

`m` is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_SCI3m_Send

Starts SCI3 data transmission.

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

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

[Syntax]

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_SCI3m_Send ( const uint8_t * tx_buf, uint16_t tx_num );
```

Remark  乽n は、is the channel number を意味します。 乿

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint8_t * tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**R_SCI3m_Receive**

Starts SCI3 data reception.

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

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

[Syntax]

```c
#include "r_cg_macrodriver.h"

MD_STATUS   R_SCI3m_Receive ( uint8_t * rx_buf, uint16_t rx_num );
```

Remark  $m$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * rx_buf;</td>
<td>Pointer to a buffer storing the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
R_SCI3m_Multiprocessor_Send

Starts SCI3 data transmission by multiprocessor mode.

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

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

**[Syntax]**

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

Remark  `m` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint8_t * tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t rx_id;</td>
<td>Reception id</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
R_SCI3m_Multiprocessor_Receive

Starts SCI3 data reception by multiprocessor mode.

Remark 1. This API function performs byte-level SCI3 reception the number of times specified by the argument \( rx\_num \), and stores the data in the buffer specified by the argument \( rx\_buf \).

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

[Syntax]

\[
\text{MD\_STATUS R_SCI3m\_Multiprocessor\_Receive( uint8\_t * rx\_buf, uint16\_t rx\_num, uint8\_t rx\_id);}
\]

Remark \( m \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer storing the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t rx_id;</td>
<td>Reception id</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
### 3.2.10 UART Interface

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

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

Performs initialization necessary to control the UART interface functions.

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_UARTm_Create_UserInit

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

**Remark**
This API function is called as the `R_UARTm_Create` callback routine.

**[Syntax]**

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

**Remark**
`m` is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
r_uartm_interrupt_receive

Performs processing in response to the UART reception interrupt.

[Syntax]

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

Remark \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_uartm_interrupt_error

Performs processing in response to the UART error interrupt.

[Syntax]

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

Remark

$m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_uartm_interrupt_send**

Performs processing in response to the UART communication interrupt.

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_uartm_callback_receiveend**

Performs processing in response to the UART reception interrupt.

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

**[Syntax]**

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

**Remark** `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_uartm_callback_sendend**

Performs processing in response to the UART communication interrupt.

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

**[Syntax]**

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

**Remark**
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_uartm_callback_error

Performs processing in response to the UART error interrupt.

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

[Syntax]

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

Remark $m$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t err_type;</td>
<td>Trigger for UART error interrupt</td>
</tr>
<tr>
<td></td>
<td>0x00x01B : Bit error</td>
<td>0x00x10B : Overrun error</td>
</tr>
<tr>
<td></td>
<td>0x01x0xB : Framing error</td>
<td>0100x0xB : Parity error</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_UARTm_Start

Sets UART communication to standby mode.

[Syntax]

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

Remark  $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### R_UARTm_Stop

Ends UART communication.

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_UARTm_Send**

Start UART data transmission.

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

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

**[Syntax]**

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

Remark  `m` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><code>const uint16_t * tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_num;</code></td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

R_UARTm_Receive

Start UART data reception.

Remark 1. This API function performs 1 byte-level UART reception the number of times specified by the argument \textit{rx\_num}, and stores the data in the buffer specified by the argument \textit{rx\_buf}.

Remark 2. Starts after this API function is called, and \textbf{R_UARTm\_Start} is then called.

**[Syntax]**

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

Remark \( m \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>const uint16_t * rx_buf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

### 3.2.11 Window Watchdog Timer

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

Table 3.11 API Functions: [Window watchdog timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_WDTm_Create</td>
<td>Performs initialization necessary to control the watchdog timer functions.</td>
</tr>
<tr>
<td>R_WDTm_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer functions.</td>
</tr>
<tr>
<td>r_wdtm_interrupt</td>
<td>Performs processing in response to the interval interrupt.</td>
</tr>
<tr>
<td>R_WDTm_Restart</td>
<td>Clears the watchdog timer counter and resumes counting.</td>
</tr>
</tbody>
</table>
3. API FUNCTIONS

R_WDTm_Create

Performs initialization necessary to control the watchdog timer functions.

[Syntax]

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

Remark  $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_WDTm_Create_UserInit

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

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

[Syntax]

\[
\text{void R_WDTm_Create_UserInit ( void );}
\]

Remark \( m \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**r_wdtm_interrupt**

Performs processing in response to the interval interrupt.

**[Syntax]**

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

Remark  

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_WDTm_Restart

Clears the watchdog timer counter and resumes counting.

#### [Syntax]

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

**Remark**  
$m$ is the channel number.

#### [Argument(s)]

None.

#### [Return value]

None.
3.2.12 OS Timer

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

Table 3.12  API Functions: [OS timer]

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

Performs initialization necessary to control the OS timer functions.

**[Syntax]**

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

Remark  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_OSTMm_Create_UserInit

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

Remark This API functions is called as the \texttt{R_OSTM\_m\_Create} callback routine.

[Syntax]

\begin{verbatim}
void R_OSTMm_create_UserInit ( void );
\end{verbatim}

Remark \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_ostmm_interrupt**

Performs processing in response to the OS timer interrupt.

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

**[Syntax]**

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

**Remark** $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### API FUNCTIONS

**R_OSTM\_m\_Start**

Start OS timer count.

**[Syntax]**

```c
void R_OSTM\_m\_Start ( void );
```

**Remark**

\(m\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_OSTMm_Stop

Stop OS timer count.

[Syntax]

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

Remark    \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
In interval timer mode, set start value of the down-counter.
In free-running comparison mode, set value for comparison.

**Syntax**

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

**Remark**

$m$ is the channel number.

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32 value;</td>
<td>Start value of the down-counter or comparison value</td>
</tr>
</tbody>
</table>

**Return value**

None.
### 3.2.13 Advanced Timer Unit IV

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_ATUIV_Common_Create</td>
<td>Performs initialization necessary to control the advanced timer unit IV functions.</td>
</tr>
<tr>
<td>R_ATUIV_Common_Create_UserInit</td>
<td>Performs user-defined initialization relating to the advanced timer unit IV functions.</td>
</tr>
<tr>
<td>R_ATUIV_Timerkn_Create</td>
<td>Performs initialization necessary to control the each timer functions.</td>
</tr>
<tr>
<td>R_ATUIV_Timerkn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the each timer functions.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_overflow_interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_interrupt</td>
<td>Performs processing in response to the timer interrupt (Input capture, compare match).</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_icrnx_interrupt</td>
<td>Performs processing in response to the ICRnx register interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_ocrnx_interrupt</td>
<td>Performs processing in response to the OCRnx register interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_tcintnx_interrupt</td>
<td>Performs processing in response to the TCNTnx interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_cmfnx_interrupt</td>
<td>Performs processing in response to the CMFnx register interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_callback_ocrc</td>
<td>Performs processing in response to the OCRCnm register interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_callback_grc</td>
<td>Performs processing in response to the GRCnm register interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_underflow_interrupt</td>
<td>Performs processing in response to the underflow interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_comparex_interrupt</td>
<td>Performs processing in response to the compare match compare register and counter interrupt</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_callback_overflow</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_callback_cycle</td>
<td>Performs processing in response to the cycle match interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_callback_duty</td>
<td>Performs processing in response to the duty match interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_fifo_overflow_interrupt</td>
<td>Performs processing in response to the FIFO overflow interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_fifo_datafull_interrupt</td>
<td>Performs processing in response to the FIFO data full interrupt.</td>
</tr>
<tr>
<td>r_atuiv_timerkmn_tcntk_overflow_interrupt</td>
<td>Performs processing in response to the counter overflow interrupt.</td>
</tr>
<tr>
<td>R_ATUIV_Timerk_OperationOn</td>
<td>Enables operation of each timer.</td>
</tr>
<tr>
<td>R_ATUIV_Timerk_OperationOff</td>
<td>Disables operation of each timer.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Start</td>
<td>Starts each timer count.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Stop</td>
<td>Ends each timer count.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_CaptureValue</td>
<td>Reads the input capture register of the timer.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Set_Compame_Match</td>
<td>Sets the compare match register.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Set_One_Shot_Pulse</td>
<td>Renewal the one shot pulse register.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Forced_Compare_Match</td>
<td>Performs forced compare match.</td>
</tr>
<tr>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Forced_Output_Compare_Match</td>
<td>Performs forced output compare match.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Start_Down_Count</td>
<td>Starts the down count.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_InputCapturex</td>
<td>Reads the input capture register of the timer.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_xpin_Output_Normal</td>
<td>Output of pin provides normal output.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_xpin_Output_Low</td>
<td>Output of pin provides low output.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_xpin_Output_High</td>
<td>Output of pin provides high output.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_Count</td>
<td>Reads the counter value of measurement.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_PWM_Measure_Value</td>
<td>Reads the PWM wave of measurement.</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Get_Measure_Value</td>
<td>Reads the value of measurement (number of edges, off-state duty cycle, PWM cycle, edge input time).</td>
</tr>
<tr>
<td>R_ATUIV_Timerknm_Reset_FIFO</td>
<td>Resets FIFO to the idle state.</td>
</tr>
</tbody>
</table>
R_ATUIV_Common_Create

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_ATUIV_Common_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_ATUIV_Timer\(kn\)\_Create**

Performs initialization necessary to control the each timer functions.

**[Syntax]**

```c
void R_ATUIV_Timer\(k\)\_Create ( void );

void R_ATUIV_Timer\(kn\)\_Create ( void );
```

Remark \(k\) is the timer kind, \(n\) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerkn_Create_UserInit

Performs user-defined initialization relating to the each timer functions.
Remark This API functions is called as the R_ATUIV_Timerkn_Create callback routine.

[Syntax]

void R_ATUIV_Timerkn_Create_UserInit ( void );

[Argument(s)]
None.

[Return value]
None.
**r_atuiv_timerknm_overflow_interrupt**

Performs processing in response to the overflow interrupt.

**[Syntax]**

```c
void r_atuiv_timerknm_overflow_interrupt ( void );

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_atuiv_timerknm_interrupt

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

**[Syntax]**

```c
void r_atuiv_timerknm_interrupt ( void );

void r_atuiv_timerkn_channelm_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_atuiv_timerknm_icrnx_interrupt**

Performs processing in response to the ICRnx register interrupt.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_atuiv_timerknnmocrnx_interrupt

Performs processing in response to the OCRnx register interrupt.

**[Syntax]**

```c
void r_atuiv_timerknnmocrnx_interrupt ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_atuiv_timerknm_tcntnx_interrupt**

Performs processing in response to the TCNTnx register interrupt.

**[Syntax]**

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

Remark  
\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number, \( x \) is the register number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_atuiv_timerknm_cmfnx_interrupt**

Performs processing in response to the CMFnx register interrupt.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_atuiv_timerknm_callback_ocrc**

Performs processing in response to the OCRC\(n\)m register interrupt.

**Remark**  This API function is called as the callback routine of interrupt process `r_atuiv_timerknm_interrupt` corresponding to the OCRC\(n\)m register interrupt.

**[Syntax]**

```c
void r_atuiv_timerknm_callback_ocrc ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_atuiv_timerknm_callback_grc

Performs processing in response to the GRCnm register interrupt.

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

[Syntax]

```c
void r_atuiv_timerknm_callback_grc ( void );
```

Remark  
\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Perform processing in response to the underflow interrupt.

[Syntax]

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

Remark  
\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_atuiv_timerknm_comparex_interrupt

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

[Syntax]

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

Remark

- \( k \) is the timer kind,
- \( n \) is the unit number,
- \( m \) is the channel number,
- \( x \) is the register number.

[Argument(s)]
None.

[Return value]
None.
r_atuiv_timerknm_callback_overflow

Performs processing in response to the overflow interrupt.

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

[Syntax]

```c
void r_atuiv_timerknm_callback_overflow_callback ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
r_atuiv_timerknm_callback_cycle

Performs processing in response to the cycle match interrupt.

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

[Syntax]

```c
void r_atuiv_timerknm_channelm_cycle_callback ( void );
```

Remark  \( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_atuiv_timerknm_callback_duty**

Performs processing in response to the duty match interrupt.

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

**Syntax**

```c
void r_atuiv_timerknm_callback_duty ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_atuiv_timerknm_fifo_overflow_interrupt

Performs processing in response to the FIFO overflow interrupt.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the FIFO data full interrupt.

**[Syntax]**

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

Remark  
\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_atuiv_timerkntc_knntcnt_k_overflow_interrupt

Performs processing in response to the counter overflow interrupt.

[Syntax]

```
void r_atuiv_timerkntc_k_overflow_interrupt ( void );
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
Enables operation of each timer.

**Syntax**

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

Remark

$k$ is the timer kind.

**Argument(s)**

None.

**Return value**

None.
Disables operation of each timer.

**[Syntax]**

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

**Remark**

\( k \) is the timer kind.

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerknm_Start

Starts each timer count.

[Syntax]

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

[Remark]

\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
[Syntax]

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

**Remark**

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

[Argument(s)]

None.

[ReturnValue]

None.
**R_ATUIV_Timerknm_Get_PulseWidth**

Reads the input pulse width of the timer.

**[Syntax]**

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

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * width;</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_ATUIV_Timerknm_Get_CaptureValue

Reads the input capture register of the timer.

[Syntax]

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

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * value;</td>
<td>Pointer to area in which to store the results of input capture register</td>
</tr>
</tbody>
</table>

[Return value]

None.
Sets the compare match register.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerknm_Set_One_Shot_Pulse

Renewal the one shot pulse register.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.
R_ATUIV_Timerknm_Forced_Compare_Match

Performs forced compare match.

[Syntax]

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

[Remark]

\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
### R_ATUIV_Timerknm_Forced_Output_Compare_Match

Performs forced output compare match.

#### Syntax

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

Remark  

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

#### Argument(s)

None.

#### Return value

None.
**R_ATUIV_Timerknm_Start_Down_Count**

Starts the down count.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerknm_Get_InputCapture

Reads the input capture register of the timer.

**[Syntax]**

```c
void    R_ATUIV_Timerknm_Get_InputCapture ( uint32_t * value );
```

**Remark**

\( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * value;</td>
<td>Pointer to area in which to store the results of input capture register</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Output of pin provides normal output.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerknm_xpin_Output_Low

Output of pin provides low output.

[Syntax]

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

Remark  

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

[Argument(s)]

None.

[Return value]

None.
Output of pin provides high output.

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ATUIV_Timerknm_Get_Count

Reads the counter value of measurement.

[Syntax]

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

Remark  

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * count;</td>
<td>Pointer to area in which to store the counter value</td>
</tr>
</tbody>
</table>

[Return value]

None.
Reads the PWM wave of measurement.

**[Syntax]**

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

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * low_width;</td>
<td>Pointer to area in which to store the results of low level width</td>
</tr>
<tr>
<td></td>
<td>uint32_t * edge_width;</td>
<td>Pointer to area in which to store the results of edge interval count</td>
</tr>
</tbody>
</table>

**[Return value]**  
None.
**R_ATUIV_Timerknm_Get_Measure_Value**

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

**[Syntax]**

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

**Remark**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * edge;</td>
<td>Pointer to area in which to store the number of edges</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * off_duty_cycle;</td>
<td>Pointer to area in which to store the off-state duty cycle</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * pwm_cycle;</td>
<td>Pointer to area in which to store the PWM cycle</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * edge_input_time;</td>
<td>Pointer to area in which to store the edge input time</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_ATUIV_Timerknm_Reset_FIFO**

Resets FIFO to the idle state.

**[Syntax]**

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

Remark: \( k \) is the timer kind, \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.14 Timer Array Unit B

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

Table 3.14 API Functions: [Timer array unit B]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TAUBn_Create</td>
<td>Performs initialization necessary to control the timer array unit Bn.</td>
</tr>
<tr>
<td>R_TAUBn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the timer array unit Bn.</td>
</tr>
<tr>
<td>r_taubn_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>R_TAUBn_Channelm_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_TAUBn_Channelm_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_TAUBn_Channelm_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
</tbody>
</table>
**R_TAUBn_Create**

Performs initialization necessary to control the timer array unit Bn.

**[Syntax]**

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

**Remark**

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAUBn_Create_UserInit**

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

**Remark**
This API function is called as the `R_TAUBn_Create` callback routine.

**[Syntax]**

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

**Remark**

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_taubn_channelm_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.
**R_TAUBn_Channelm_Start**

Starts the count for channel \( m \).

**[Syntax]**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
ends the count for channel m.

[syntax]

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

[remark] 
n is the unit number, m is the channel number

[argument(s)]

None.

[return value]

None.
**R_TAUBn_Channelm_Get_PulseWidth**

Reads the input pulse width of the timer.

**[Syntax]**

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

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * width;</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
### 3.2.15 Timer Array Unit D

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TAUDn_Create</td>
<td>Performs initialization necessary to control the timer array unit Dn.</td>
</tr>
<tr>
<td>R_TAUDn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the timer array unit Dn.</td>
</tr>
<tr>
<td>r_taudn_channelsm_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>R_TAUDn_Channelm_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_TAUDn_Channelm_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_TAUDn_Channelm_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
</tbody>
</table>
**R_TAUDn_Create**

Performs initialization necessary to control the timer array unit Dn.

**[Syntax]**

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

Remark: \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAUDn_Create_UserInit**

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

Remark  This API function is called as the `R_TAUDn_Create` callback routine.

**[Syntax]**

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

Remark  \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return Value]**

None.
**r_taudn_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAUDn_Channelm_Start**

Starts the count for channel \( m \).

**[Syntax]**

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

**Remark**

\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAUDn_Channelm_Stop**

Ends the count for channel m.

**[Syntax]**

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

Remark  
\( n \) is the unit number, \( m \) is the channel number

**[Argument(s)]**

None.

**[Return value]**

None.
R_TAUDn_Channelm_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

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

Remark

\( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * width;</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

[Return value]

None.
### 3.2.16 Timer Array Unit J

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TAUJn_Create</td>
<td>Performs initialization necessary to control the timer array unit Jn.</td>
</tr>
<tr>
<td>R_TAUJn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the timer array unit Jn.</td>
</tr>
<tr>
<td>r_taujn_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>R_TAUJn_Channelm_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_TAUJn_Channelm_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_TAUJn_Channelm_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
</tbody>
</table>
**R_TAUJn_Create**

Performs initialization necessary to control the timer array unit Jn.

**[Syntax]**

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

**Remark**

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAUJn_Create_UserInit**

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

Remark   This API function is called as the `R_TAUJn_Create` callback routine.

**[Syntax]**

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

Remark   n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_taujn_channelm_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_taujn_channelm_interrupt ( void );
```

Remark  

$n$ is the unit number, $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_TAUJn_Channelm_Start**

Starts the count for channel \( m \).

**[Syntax]**

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

Remark  
\( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_TAUJn_Channelm_Stop

Ends the count for channel m.

[Syntax]

void  R_TAUJn_Channelm_Stop ( void );

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

[Argument(s)]

None.

[Return value]

None.
**R_TAUIjn_Channelm_Get_PulseWidth**

Reads the input pulse width of the timer.

### Syntax

```c
void R_TAUIjn_Channelm_Get_PulseWidth ( uint32_t * width );
```

### Remark

$n$ is the unit number, $m$ is the channel number.

### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * width;</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

### Return value

None.
### 3.2.17 Timer Option

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

Table 3.17 API Functions: [Timer option]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TAPAm_Create</td>
<td>Performs initialization necessary to control the timer option functions.</td>
</tr>
<tr>
<td>R_TAPAm_Create_UserInit</td>
<td>Performs user-defined initialization relating to the timer option functions.</td>
</tr>
<tr>
<td>R_TAPAm_Start</td>
<td>Enables asynchronous Hi-Z control.</td>
</tr>
<tr>
<td>R_TAPAm_Stop</td>
<td>Disables asynchronous Hi-Z control.</td>
</tr>
<tr>
<td>R_TAPAm_Trigger_Start</td>
<td>Sets the Hi-Z control signal to the low level.</td>
</tr>
<tr>
<td>R_TAPAm_Trigger_Stop</td>
<td>Sets the Hi-Z control signal to the high level.</td>
</tr>
</tbody>
</table>
**R_TAPAm_Create**

Performs initialization necessary to control the timer option functions.

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAPA\_m\_Create\_UserInit**

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

Remark This API function is called as the R_TAPA\_m\_Create callback routine.

**[Syntax]**

```c
void  R_TAPA\_m\_Create\_UserInit ( void );
```

Remark \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TAPAm_Start**

Enables asynchronous Hi-Z control.

**[Syntax]**

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

Remark  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables asynchronous Hi-Z control.

**[Syntax]**

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

**Remark**

\( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
API FUNCTIONS

R_TAPAm_Trigger_Start

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

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Sets the Hi-Z control signal to the high level.

[Syntax]

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

Remark  

$m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.18 Peripheral Interconnection

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_PICn_Create</td>
<td>Performs initialization necessary to control the peripheral interconnection functions.</td>
</tr>
<tr>
<td>R_PICn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the peripheral interconnection functions.</td>
</tr>
<tr>
<td>R_PICn_Timer_SyncStart</td>
<td>Generates start triggers of the timers for which simultaneous start is enabled.</td>
</tr>
</tbody>
</table>
**R_PICn_Create**

Performs initialization necessary to control the peripheral interconnection functions.

**[Syntax]**

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

**Remark**

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_PICn_Create_UserInit**

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

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

**[Syntax]**

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

**Remark**  $n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Generates start triggers of the timers for which simultaneous start is enabled.

**[Syntax]**

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

Remark  
\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.19 A/D converter

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

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

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

[Syntax]

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

Remark  
\( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_ADCn_Create_UserInit

Performs user-defined initialization relating to the A/D converter functions.
Remark   This API function is called as the R_ADCn_Create callback routine.

Syntax

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

Remark   $n$ is the unit number.

Argument(s)

None.

Return value

None.
r_adcn_error_interrupt

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

[Syntax]

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

Remark

\[n\] is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_adc{n}_scan_group{m}_end_interrupt

Performs processing in response to the scan group end interrupt.

**[Syntax]**

```c
void r_adc{n}_scan_group{m}_end_interrupt ( void );
```

**Remark**

\( n \) is the unit number, \( m \) is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
Perform processing in response to the MPX interrupt.

**[Syntax]**

```c
void r_adc{n}_multiplexer_request_interrupt ( void );
```

**Remark**

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_adc_n_adc_summation_channel_m_end_interrupt

Performs processing in response to the accumulation end interrupt.

[Syntax]

```
void r_adc_n_adc_summation_channel_m_end_interrupt ( void );
```

Remark  

\( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_ADCn_Halt

Halts A/D converter.

[Syntax]

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

Remark $n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_ADCn_SetMultiplexerCommand

Sets MPX command.

[Syntax]

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

Remark  

$n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_ADCn_ScanGroupm_Start

Starts ADC scan group conversion.

[Syntax]

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

[Remark]

\( n \) is the unit number, \( m \) is the scan group number.

[Argument(s)]

None.

[Return value]

None.
**R_ADCn_ScanGroupm_GetResult**

Reads the results of ADC scan group conversion.

**[Syntax]**

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

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * buffer;</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_ADCn_ScanGroupm_GetFloatingPointDataResult**

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

**[Syntax]**

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

**Remark**

$n$ is the unit number, $m$ is the scan group number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * buffer;</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Starts ADC scan group timer.

**Syntax**

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

**Remark**

$n$ is the unit number, $m$ is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
Ends ADC scan group timer.

**Syntax**

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

**Remark**

$n$ is the unit number, $m$ is the scan group number.

**Argument(s)**

None.

**Return value**

None.
# R_ADCn_ADCSummation_Channelm_GetResult

Reads the results of accumulation data.

## Syntax

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

### Remark

$n$ is the unit number, $m$ is the channel number.

## Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * buffer</td>
<td>Pointer to area in which to store the results of accumulation data</td>
</tr>
</tbody>
</table>

## Return value

None.
### R_ADCn_ADCSummation_Start

Starts summation function.

**[Syntax]**

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

**Remark**

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_ADCn_ADCSummation_Stop

Ends summation function.

**[Syntax]**

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

*Remark*  
\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_ADC_SyncStart

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

### Syntax

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

### Argument(s)

None.

### Return value

None.
3. API FUNCTIONS

R_ADC_SyncTimerStart

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_ADCn_ScanGroupm_OperationOn

Starts ADC scan group scan.

[Syntax]

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

Remark

\( n \) is the unit number, \( m \) is the scan group number.

[Argument(s)]

None.

[Return value]

None.
### R_ADCn_TH_Groupx_Start

Starts ADC T&H group hold.

**[Syntax]**

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

**Remark**

$n$ is the unit number, $x$ is the group number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_ADCn_TH_Sampling_Start**

Starts ADC T&H sampling.

**[Syntax]**

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

**Remark**

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.20 Delta-Sigma AD converter

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DSADC_Create</td>
<td>Performs initialization necessary to control the $\Delta \Sigma$ A/D converter functions.</td>
</tr>
<tr>
<td>R_DSADC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the $\Delta \Sigma$ A/D converter functions.</td>
</tr>
<tr>
<td>r_dsadc_error_interrupt</td>
<td>Performs processing in response to the $\Delta \Sigma$ A/D error interrupt.</td>
</tr>
<tr>
<td>R_DSADC_SYNCStart</td>
<td>Starts $\Delta \Sigma$ AD conversion set as enable AD synchronization start.</td>
</tr>
<tr>
<td>R_DSADCm_Start</td>
<td>Starts $\Delta \Sigma$ A/D conversion.</td>
</tr>
<tr>
<td>R_DSADCm_Stop</td>
<td>Ends $\Delta \Sigma$ A/D conversion.</td>
</tr>
<tr>
<td>R_DSADCm_GetResult</td>
<td>Reads the results of $\Delta \Sigma$ A/D conversion.</td>
</tr>
</tbody>
</table>
R_DSADC_Create

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_DSADC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
Performs processing in response to the ΔΣ A/D error interrupt.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### R_DSADC_SyncStart

Starts ΔΣ AD conversion set as enable AD synchronization start.

#### [Syntax]

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

#### [Argument(s)]

None.

#### [Return value]

None.
## R_DSADCm_Start

Starts ΔΣ A/D conversion.

### Syntax

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

### Remark

$m$ is the channel number.

### Argument(s)

None.

### Return value

None.
R_DSADCm_Stop

Ends ΔΣ A/D conversion.

**[Syntax]**

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

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DSADCm_GetResult**

Reads the results of ΔΣ A/D conversion.

**[Syntax]**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * result;</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
### 3.2.21 Digital Filter

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

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

Performs initialization necessary to control the digital filter functions.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_DFE_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
r_dfe_error_interrupt

Performs processing in response to the DFE error interrupt.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
### R_DFE_Set_SoftwareData

Sets the data to be processed in a target filter.

#### Syntax

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

#### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t tag;</td>
<td>Set the same value as the channel tag</td>
</tr>
<tr>
<td>I</td>
<td>int16_t const data;</td>
<td>Software input data</td>
</tr>
</tbody>
</table>

#### Return value

None.
R_DFE_Generate_SoftwareTrigger

Generates the software trigger.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
### R_DFE_Channelm_Create

Performs initialization necessary to control the digital filter channel functions.

**Syntax**

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

**Remark**

$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DFE_Channelm_Create_UserInit**

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

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

**[Syntax]**

```c
void R_DFE_Channelm_Create_UserInit ( void );
```

Remark       *m* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_dfe_channel\texttt{m} \_interrupt

Performs processing in response to the DFE channel interrupt.

**[Syntax]**

```c
void r_dfe_channel\texttt{m} \_interrupt ( void );
```

Remark \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**AP4 3. API FUNCTIONS**

**r_dfe_channelm_callback_output_data**

Performs processing in response to the output data interrupt.

**Remark** This API function is called as the `r_dfe_channelm_interrupt` callback routine.

**[Syntax]**

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

**Remark** \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_dfe_channelm_callback_condition_match

Performs processing in response to the condition match interrupt.

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

[Syntax]

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

Remark \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_DFE_Channel\(m\)_Enable

Enables the DFE channel.

[Syntax]

```c
void R_DFE_Channel\(m\)_Enable ( void );
```

Remark \(m\) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Disables the DFE channel.

**[Syntax]**

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

Remark  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DFE_Channel\textit{m}_GetResult**

Reads the results of DFE processing calculation.

**[Syntax]**

```c
void R_DFE_Channel\textit{m}_GetResult ( int32_t * \textit{buffer} );
```

**Remark**

\textit{m} is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>int32_t * \textit{buffer};</td>
<td>Pointer to area in which to store the results of DFE processing calculation</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
3.2.22 Data CRC

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

Table 3.22 API Functions: [Data CRC]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DCRAn_Create</td>
<td>Performs initialization necessary to control the data CRC functions.</td>
</tr>
<tr>
<td>R_DCRAn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data CRC functions.</td>
</tr>
<tr>
<td>R_DCRAn_Input32bitData</td>
<td>Sets the calculation data for 32 bit width.</td>
</tr>
<tr>
<td>R_DCRAn_Input16bitData</td>
<td>Sets the calculation data for 16 bit width.</td>
</tr>
<tr>
<td>R_DCRAn_Input8bitData</td>
<td>Sets the calculation data for 8 bit width.</td>
</tr>
<tr>
<td>R_DCRAn_GetResult_32bitData</td>
<td>Reads the results of CRC calculation for 32 bit width.</td>
</tr>
<tr>
<td>R_DCRAn_GetResult_16bitData</td>
<td>Reads the results of CRC calculation for 16 bit width.</td>
</tr>
</tbody>
</table>
### R_DCRA\_Create

Performs initialization necessary to control the data CRC functions.

**[Syntax]**

```c
void R_DCRA\_Create ( void );
```

**Remark**

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs user-defined initialization relating to the data CRC functions.

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

**[Syntax]**

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

Remark  \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Sets the calculation data for 32 bit width.

**[Syntax]**

```c
void R_DCRA\_n\_Input32bitData ( const uint32\_t* data, uint32\_t data\_num );
```

**Remark**  
$n$ is the unit number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint32_t* data;</td>
<td>Pointer to a buffer storing the calculation data</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Sets the calculation data for 16 bit width.

**[Syntax]**

```c
void R_DCRA_n_Input16bitData ( const uint16_t * data, uint32_t data_num );
```

Remark  

$n$ is the unit number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint16_t * data;</td>
<td>Pointer to a buffer storing the calculation data</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_DCRAn_Input8bitData

Sets the calculation data for 8 bit width.

[Syntax]

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

Remark  

\( n \) is the unit number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint8_t * data;</td>
<td>Pointer to a buffer storing the calculation data</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_DCRAn_GetResult_32bitData

Reads the results of CRC calculation for 32 bit width.

[Syntax]

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

Remark

\( n \) is the unit number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * data;</td>
<td>Pointer to area in which to store the results of calculation data</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_DCRAn_GetResult_16bitData**

Reads the results of CRC calculation for 16 bit width.

**[Syntax]**

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

**Remark**

$n$ is the unit number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * data;</td>
<td>Pointer to area in which to store the results of calculation data</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
3.2.23 Real-Time Clock

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_RTC_Create</td>
<td>Performs initialization necessary to control the real-time clock.</td>
</tr>
<tr>
<td>R_RTC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the real-time clock.</td>
</tr>
<tr>
<td>R_RTC_Start</td>
<td>Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>R_RTC_Stop</td>
<td>Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>R_RTC_Set_HourSystem</td>
<td>Sets the clock type (12-hour or 24-hour clock) of the real-time clock.</td>
</tr>
<tr>
<td>R_RTC_Set.CounterValue</td>
<td>Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.</td>
</tr>
<tr>
<td>R_RTC_Get.CounterValue</td>
<td>Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.</td>
</tr>
<tr>
<td>R_RTC_Set.AlarmOn</td>
<td>Starts the alarm interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set.AlarmOff</td>
<td>Ends the alarm interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set.AlarmValue</td>
<td>Sets the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>R_RTC_Get.AlarmValue</td>
<td>Reads the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>R_RTC_Set.ConstPeriodInterruptOn</td>
<td>Sets the cycle of the periodic interrupts, then starts the periodic interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set.ConstPeriodInterruptOff</td>
<td>Ends the periodic interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set.1secondInterruptOn</td>
<td>Starts the 1 second interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set.1secondInterruptOff</td>
<td>Ends the 1 second interrupt function.</td>
</tr>
<tr>
<td>R_RTC_Set_RTC1HZOn</td>
<td>Enables output of the correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>R_RTC_Set_RTC1HZOff</td>
<td>Disables output of the correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>r_rtc_interrupt_periodic</td>
<td>Performs processing in response to the periodic interrupt.</td>
</tr>
<tr>
<td>r_rtc_interrupt_alarm</td>
<td>Performs processing in response to the alarm interrupt.</td>
</tr>
<tr>
<td>r_rtc_interrupt_1second</td>
<td>Performs processing in response to the 1 second interrupt.</td>
</tr>
</tbody>
</table>
R_RTC_Create

Performs initialization necessary to control the real-time clock.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_RTC_Create_UserInit

Performs user-defined initialization relating to the real-time clock.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_RTC_Stop

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

[Syntax]

    void    R_RTC_Stop ( void );

[Argument(s)]

    None.

[Return value]

    None.
R_RTC_Set_HourSystem

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

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_hour_system_t hour_system;</td>
<td>Clock type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HOUR12: 12-hour clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HOUR24: 24-hour clock</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before change to setting)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

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

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

[Syntax]

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

[Argument(s)]

I/O | Argument | Description
--- | --- | ---
I | rtc_counter_value_t counter_write_val; | Counter value

Remark

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

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

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before change to setting)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting)</td>
</tr>
</tbody>
</table>

Remark

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

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

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_counter_value_t * const counter_read_val;</td>
<td>Pointer to structure in which to store the counter value being read</td>
</tr>
</tbody>
</table>

Remark: See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before reading)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after reading)</td>
</tr>
</tbody>
</table>

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

Starts the alarm interrupt function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
Ends the alarm interrupt function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Set_AlarmValue**

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

**Syntax**

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

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_alarm_value_t alarm_val;</td>
<td>Alarm conditions (weekday, hour, minute)</td>
</tr>
</tbody>
</table>

**Remark**

Below is shown the structure rtc_alarm_value_t (alarm conditions).

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

- **alarmwm (Minute)**
  
  Below are shown the meanings of each bit of the structure member alarmwm.
  
<table>
<thead>
<tr>
<th>7</th>
<th>4</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  BCD code (minute: digit 10)
  BCD code (minute: digit 1)
  0: Fixed

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

<table>
<thead>
<tr>
<th>0: AM</th>
<th>1: PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

  BCD code (hour: digit 1)
  BCD code (hour: digit 10)
  00: Fixed

- **alarmww (Weekday)**
  
  Below are shown the meanings of each bit of the structure member alarmww.
[Return value]

None.
**R_RTC_Get_AlarmValue**

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

**[Syntax]**

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

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_alarm_value_t * const alarm_val;</td>
<td>Pointer to structure in which to store the conditions being read</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_RTC_Set_ConstPeriodInterruptOn

Sets the cycle of the periodic interrupts, then starts the periodic interrupt function.

[Syntax]

```c
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"

MD_STATUS   R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| I   | rtc_int_period_t  period; | Interrupt INTRTC cycle  
QUARTERSEC: 0.25 seconds  
HALFSEC: 0.5 seconds  
ONESEC: 1 second  
ONEMIN: 1 minute  
ONEHOUR: 1 hour  
ONEDAY: 1 day  
ONEMONTH: 1 month |

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Ends the periodic interrupt function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Set_1secondInterruptOn**

Starts the 1 second interrupt function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Set_1secondInterruptOff**

Ends the 1 second interrupt function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Set_RTC1HZOn**

Enables output of the correction clock (1 Hz).

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Set_RTC1HZOff**

Disables output of the correction clock (1 Hz).

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_rtc_interrupt_periodic

Performs processing in response to the periodic interrupt.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
r_rtc_interrupt_alarm

Performs processing in response to the alarm interrupt.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
The function `r_rtc_interrupt_1second` performs processing in response to the 1-second interrupt. The function is called as the interrupt process corresponding to the 1-second interrupt.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.24 Key Return

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

Table 3.24 API Functions: [Key Return]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_KEY_Create</td>
<td>Performs initialization necessary to control the key return functions.</td>
</tr>
<tr>
<td>R_KEY_Create_UserInit</td>
<td>Performs user-defined initialization relating to the key return functions.</td>
</tr>
<tr>
<td>r_key_interrupt</td>
<td>Performs processing in response to the key return interrupt.</td>
</tr>
<tr>
<td>R_KEY_Start</td>
<td>Enables the acceptance of the key return interrupt.</td>
</tr>
<tr>
<td>R_KEY_Stop</td>
<td>Disables the acceptance of the key return interrupt.</td>
</tr>
</tbody>
</table>
**R_KEY_Create**

Performs initialization necessary to control the key return functions.

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
**R_KEY_Create_UserInit**

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

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

**[Syntax]**

```c
void R_KEY_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**r_key_interrupt**

Performs processing in response to the key return interrupt.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_KEY_Start

Enables the acceptance of the key return interrupt.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_KEY_Stop

Disables the acceptance of the key return interrupt.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.25 Stand-By Controller

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

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

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

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
R_STBC_Prepare_Stop_Mode

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_STBC_Start_Deep_Stop_Mode**

Start stand-by (DeepSTOP mode).

**Remark**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_STBC_Prepare_Deep_Stop_Mode

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_STBC_Deep_Stop_Loop**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_STBC_Prepare_Stop_Mode_Set_Peripheral

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_STBC_Prepare_Stop_Mode_Set_Interrupt**

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

**[Syntax]**

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

**Argument(s)**

None.

**Return value**

None.
**R_STBC_Prepare_Stop_Mode_Set_Clock_Mask**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_STBC_Prepare_Stop_Mode_Set_Clock_Source

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
### R_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_STBC_Prepares_Deep_Stop_Mode_Set_Interrupt**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep 01, 2015</td>
<td>-</td>
<td>First Edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Mar 01, 2016</td>
<td>38-40</td>
<td>3.2.4 Interrupt API addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>71-83</td>
<td>3.2.7 Clock Serial Interface G Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120-132</td>
<td>3.2.10 UART Interface Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>186-192</td>
<td>3.2.14 Timer Array Unit B Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>193-199</td>
<td>3.2.15 Timer Array Unit D Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200-206</td>
<td>3.2.16 Timer Array Unit J Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>237-239</td>
<td>3.2.19 A/D converter API addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>270-291</td>
<td>3.2.23 Real-Time Clock Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>292-297</td>
<td>3.2.24 Key Return Chapter addition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>298-309</td>
<td>3.2.25 Stand-By Controller Chapter addition</td>
</tr>
</tbody>
</table>